I Workshop en Deep Learning (DeepL)

SIHAM TABIK
JUAN MANUEL GÓRRIZ
ANTONIO BAHAMONDE

I Workshop en Deep Learning (DeepL)

SESIÓN 1

Optimización de las técnicas de Transfer Learning para la clasificación de la calidad estética en fotografía

Fernando Rubio
SIMD Lab, I3A
UCLM
Albacete, España
Email: fernando.rubio@uclm.es

M. Julia Flores
Departamento de Sistemas Informáticos
UCLM
España
Email: julia.flores@uclm.es

Jose M. Puerta
Departamento de Sistemas Informáticos
UCLM
España
Email: jose.puerta@uclm.es

***Abstract*—La evaluación automática de la calidad estética es un problema de visión por ordenador que consiste en cuantificar el atractivo de una fotografía. Esto es especialmente útil en las redes sociales, donde la cantidad de imágenes que se generan cada día requieren de la automatización para su procesamiento.**

Aunque ha habido progresos notables en la investigación de este campo, aún es difícil encontrar soluciones aplicables. Con este trabajo buscamos la optimización de las soluciones más prometedoras basadas en Transfer Learning. Para ello, hemos reducido la complejidad de las redes neuronales propuestas manteniendo los resultados obtenidos, mediante técnicas de "finetuning" sobre redes pre-entrenadas.

***Index Terms*—Deep Learning, Transfer Learning, Finetuning, Classification, Computer Vision**

I. Introducción

El campo de visión por ordenador es uno de los más activos en la comunidad científica debido a la gran cantidad de aplicaciones que tiene como la robótica y la seguridad. En los últimos años, las redes neuronales profundas han permitido resolver problemas, que hasta hace poco parecían inabordables. Esto ocurre también con el problema de la calidad estética.

El concepto de calidad estética en la fotografía hace referencia a las propiedades de las imágenes que las hacen atractivas o "bonitas" para la mayoría de la gente, como pueden ser los filtros aplicados, armonía de los colores, etc. No hay que confundir con la calidad de una imagen en términos de resolución. Se trata de uno de los problemas más complejos dentro del campo de visión debido a la subjetividad de la tarea, ya que la opinión de diferentes personas sobre la calidad de una única imagen puede ser muy distinta. Incluso entre expertos de fotografía puede haber opiniones diferentes.

A pesar de su dificultad, se trata de un problema que ha visto incrementado su interés enormemente debido a la gran cantidad de imágenes que se generan continuamente con las redes sociales. La automatización de esta tarea tiene aplicaciones muy interesantes como la ordenación de álbumes de imágenes en base a su calidad, especialmente útil para sitios como Flickr o Instagram. Pero también se puede utilizar para recomendaciones de filtros o incluso para evaluaciones online en una cámara, para mejorar la calidad de las fotografías que tomamos. Otro de los campos de aplicación es la publicidad, con la creación de imágenes más atractivas.

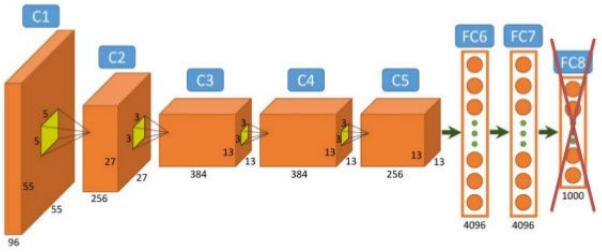


Fig. 1. Extracción de características de una red neuronal.

Tradicionalmente, la automatización de la evaluación de la calidad estética se centra en resolver un problema de clasificación binaria, donde las imágenes son clasificadas como "snapshots" (mala calidad) o "professional shots" (buena calidad). El uso del Deep Learning mediante las redes convolucionales ha mejorado los resultados en los últimos años e incluso ha permitido utilizar las probabilidades generadas en la última capa como indicadores más precisos de la calidad de la imagen.

En este artículo nos centramos en una de las estrategias más utilizadas recientemente en Deep Learning, Transfer Learning. Este concepto se basa en utilizar redes neuronales pre-entrenadas con otros datasets y aplicar dicho conocimiento a nuestro problema.

Actualmente, dos son las técnicas principales de Transfer Learning. La primera de ellas, puede verse en la Fig. 1 y que consiste en la extracción de características de una red neuronal, lo que se conoce como **ConvNet features** o **DeCaf** [1]. En este caso, obviamos la salida de la red y obtenemos las activaciones que se producen en las capas anteriores, para utilizarlas como *inputs* en otros modelos.

La segunda es el concepto de **finetuning**, que puede verse en la Fig. 2. Esta técnica de Transfer Learning consiste en la modificación de la (o las) última(s) capa(s) para ajustar

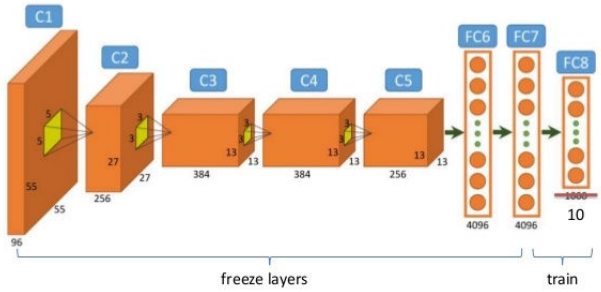


Fig. 2. Proceso de *finetuning*, donde se ha modificado la última capa de una red pre-entrenada, cuyos pesos van a ser aprendidos. El resto de capas no sufre modificaciones.

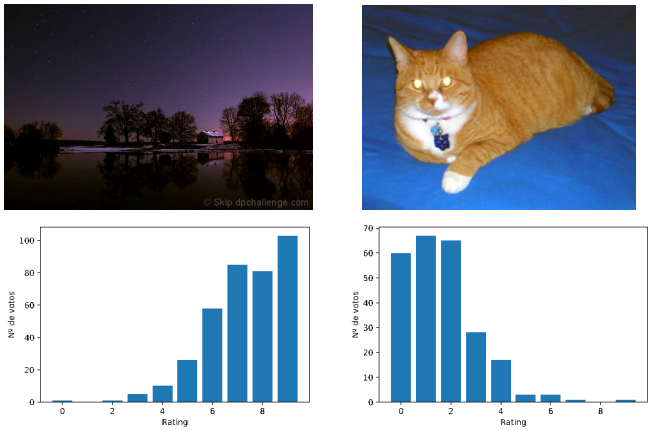


Fig. 3. Ejemplo de una imagen profesional (arriba, izquierda) y de una de mala calidad (arriba, derecha). En la parte inferior se muestra la distribución de los votos para cada imagen.

la salida de una red a nuestro problema. El siguiente paso consiste en el entrenamiento de los pesos sólo de las capas que han sido modificadas. Esto reduce el tiempo y la cantidad de datos necesarios para entrenar la red.

Cuando hablamos de Transfer Learning en imágenes, existen diferentes modelos pre-entrenados que podemos utilizar como son AlexNet [2], VGG [3], Inception [4], ResNet [5] o MobileNet [6]. Todos estos comparten una serie de propiedades y la base de datos con la que fueron entrenadas, ImageNet [7]. Este dataset consiste en un conjunto de más de 2 millones de imágenes de objetos (valla, barco, avión), animales (perro, gato, tortuga) o conceptos (atardecer, paisaje) que han sido etiquetadas con 1 de 1000 posibles "tags". Por lo tanto, ImageNet es un problema de clasificación donde la clase puede tomar 1000 posibles valores. El objetivo de utilizar Transfer Learning sobre las redes aprendidas para este problema, es aprovechar todo el conocimiento generado por una red capaz de identificar 1000 conceptos diferentes en imágenes.

Los mejores resultados obtenidos para la clasificación de la calidad estética vienen de propuestas de Transfer Learning sobre dichos modelos, especialmente con *finetuning* [8]. Sin embargo, en el estudio de estas soluciones, por lo general, sólo la última capa es modificada para el reentrenamiento. En este trabajo proponemos realizar un proceso de *finetuning* de más capas, con el objetivo de reducir el tamaño de la red, pero sin afectar a los resultados obtenidos. Esto permitirá que las redes neuronales puedan utilizarse en dispositivos más limitados en recursos computacionales como son los dispositivos móviles o las cámaras fotográficas, permitiendo la creación de aplicaciones reales.

II. Estado del arte

Las primeras aproximaciones para la evaluación de la calidad estética consisten en una clasificación binaria de las imágenes en "snapshots" o "professional shots". Sin embargo, al tratarse de un problema con tanta incertidumbre y donde la clase no está bien definida, las principales bases de datos están compuestas por imágenes donde un grupo de individuos han asignado unos ratings o votos [9]. En la Fig. 3, podemos observar dos imágenes con su distribución de votos.

Generalmente, a partir de los ratings asignados a una imagen, obtenemos estadísticos que nos permiten convertir el problema en uno de clasificación binaria. Por ejemplo, podemos utilizar métricas como la media o la mediana para separar las imágenes en "snapshots" o "professional shots". Normalmente, se fija un umbral en el punto medio del rango de los ratings, es decir, si los votos van en una escala del 1 al 10, el corte se establece en 5. Finalmente, se obtiene la media para cada imagen μ_i a partir de sus votos y se comparan con el umbral. En la Fig 3, en el caso de la izquierda, tenemos una $\mu = 8.31$, y en la derecha $\mu = 2.62$. Al tener un rango de 10 posibles valores, el umbral se situa en el 5, por lo que la imagen de la izquierda sería clasificada como "professional shot" y la de la derecha como "snapshot".

[10] y [11] proponen resolver el problema con características hechas a mano de bajo nivel para tratar de identificar propiedades más complejas de las fotografías y que sean capaces de separar ambas categorías. Sin embargo, estas propuestas pronto fueron superadas por técnicas generales de extracción de características, como GIST o SIFT [12].

Con la aparición del dataset AVA [13], el problema de la calidad estética ya contaba con una considerable base de datos de imágenes que permitía el aprendizaje de redes de Deep Learning como en [14]. Sin embargo, los resultados de entrenar una red neuronal desde cero, para este problema concreto, no han sido tan relevantes como en otros campos de visión.

[8] demuestra que realizando el proceso de *finetuning* en la última capa de los modelos AlexNet y VGG es posible obtener unos resultados más fiables que los presentados hasta ese momento. [15] también hace uso de esta técnica de Transfer Learning para predecir directamente la distribución de los votos, donde la capa de *output* tiene un tamaño de 10, correspondiente al rango de ratings de AVA, en vez de 2 de la clasificación binaria. En este último caso, en vez de *softmax* como función de activación de la última capa, utilizan Earth Mover's Distance (EMD), que obtiene la distribución de probabilidad acumulada para la imagen.

Hay que destacar que la mayoría de los trabajos sufrían de un problema con la evaluación, como se indica en [16], ya que la única métrica utilizada en muchas propuestas para validar los modelos era la tasa de aciertos o *accuracy*. En AVA, esta métrica es poco informativa, ya que si binarizamos la clase a partir de los votos, cogiendo como umbral el 5 de media (ya que el rango va de 1 a 10), observamos un desbalanceo de la clase, donde el 70% de los casos son "professional shots" y el 30% "snapshots". En este caso, reportar una tasa de aciertos del 70% no tiene valor, ya que se pueden estar clasificando todas las imágenes como buenas.

Para resolver esta situación, [16], [8] y [15] hacen uso de diferentes métricas como son el *balanced accuracy* que tiene en cuenta la tasa de aciertos por cada una de las clases o el valor AUC (Area Under the Curve), que relaciona la tasa de Verdaderos Positivos con la tasa de Falsos Positivos. En este trabajo utilizaremos esas métricas cuando trabajemos con problemas desbalanceados.

Es frecuente encontrar en la literatura propuestas que tienden a reducir las bases de datos originales en subconjuntos. Principalmente se eliminan imágenes cuya media de votos se encuentra cerca del umbral de corte. [14] y [12] utilizan una δ, de forma que si consideramos 5 el punto de corte, las imágenes $< 5 - \delta$ son consideradas "snapshots" y las $> 5 + \delta$ son etiquetadas como "professional shots", descartando el resto. Otros, como [17] o [18] seleccionan el 10% mejor y peor valoradas y el resto no se tienen en cuenta.

Es comprensible tratar de dar más peso a aquellas imágenes más informativas y tratar de reducir el ruido que pueden generar los casos cercanos a la frontera de decisión. Sin embargo, en algunas de estas propuestas, los resultados presentados han eliminado del conjunto de evaluación las imágenes, con el fin de simplificar el problema. Consideramos que estos resultados no son válidos para un escenario real, donde la mayoría de las imágenes se encuentran cerca del umbral de corte. La eliminación de imágenes sólo debe realizarse en los conjuntos de entrenamiento.

Por último, en las Fig.4 y 5 se muestran resultados preliminares de la evaluación de la calidad estética en AVA utilizando características generales de la imagen. En ambas se refleja la evolución de los resultados en base al valor δ donde sólo se descartan las imágenes del conjunto de entrenamiento. Se observa que eliminar las imágenes cercanas al punto de corte no afecta de forma significativa a los resultados, he incluso en el caso del AUC vemos un peor comportamiento. Esto corrobora lo resultados de [14], donde la reducción de las imágenes también perjudica a los modelos de Deep Learning. Por estos motivos, en este trabajo utilizaremos las bases de datos completas.

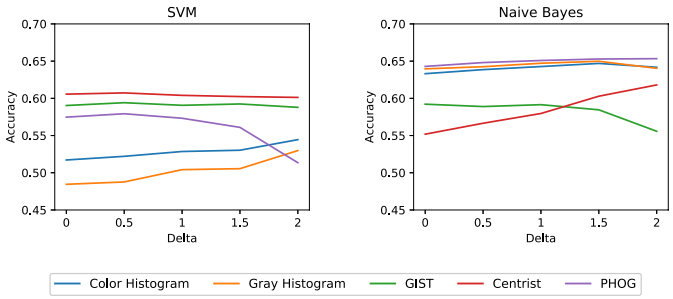


Fig. 4. Evolución del *accuracy* (tasa de aciertos) en base a la δ (Delta) para la reducción de imágenes de la base de datos AVA.

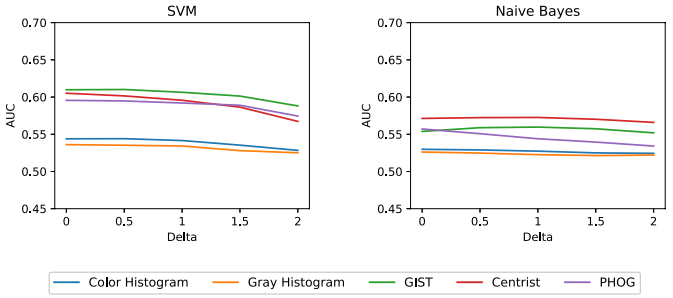


Fig. 5. Evolución del AUC en base a la δ (Delta) para la reducción de imágenes de la base de datos AVA.

III. Transfer Learning

A. Dataset

Actualmente, la base de datos referente para el tratamiento de la calidad estética es AVA. Este dataset esta compuesto por cerca de 250.000 imágenes pertenecientes a una página de retos fotográficos, DPChallenge. Cada foto ha sido valorada del 1 al 10 por diferentes usuarios, siendo el 1 que la foto es muy mala y 10 que es muy buena. Además, cada imagen viene etiquetada con el tipo de reto en la que se subió, el estilo fotográfico (si tiene alguno) y ciertas etiquetas sobre objetos que aparecen en la imagen. En los trabajos previos, esta base de datos es particionada en train y test, siendo este último de unas 20k imágenes y las 230k restantes para entrenamiento. En este trabajo utilizaremos la misma partición.

B. Extracción de ConvNet features

Durante la evaluación de una imagen en una red neuronal (proceso *forward*), no sólo se obtienen los valores de salida, en cada capa de la red se generan una serie de activaciones que pueden ser extraídas. Aunque es difícil interpretar esta información, puede utilizarse como vectores de características de la imagen, ya que tienen una gran capacidad descriptiva. Esto es lo que se conoce como *ConvNet features* y se utilizan generalmente en problemas donde no tenemos suficientes datos como para entrenar una red neuronal. Estas características son extraídas de redes pre-entrenadas y se utilizan para aprender otros modelos de clasificación.

En [16] se realiza un estudio del rendimiento de las *ConvNet features* extraídas de dos redes neuronales (AlexNet y ResNet) en la evaluación de la calidad estética. En las Fig. 6 y 7 se observan los resultados, donde los modelos entrenados con las *ConvNet features* superaban a los clasificadores que utilizaban descriptores generales de la imagen. Sin embargo, debido a la gran cantidad de características que se obtienen de las redes neuronales, no todos los modelos se muestran en este estudio.

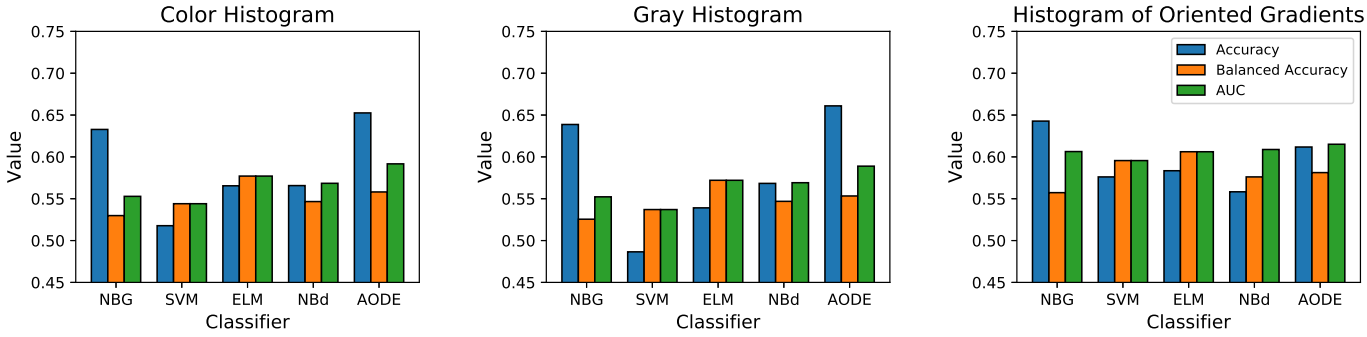


Fig. 6. Resultados de 5 clasificadores entrenados con descriptores generales de la imagen en términos de *accuracy*, *balanced accuracy* y AUC.

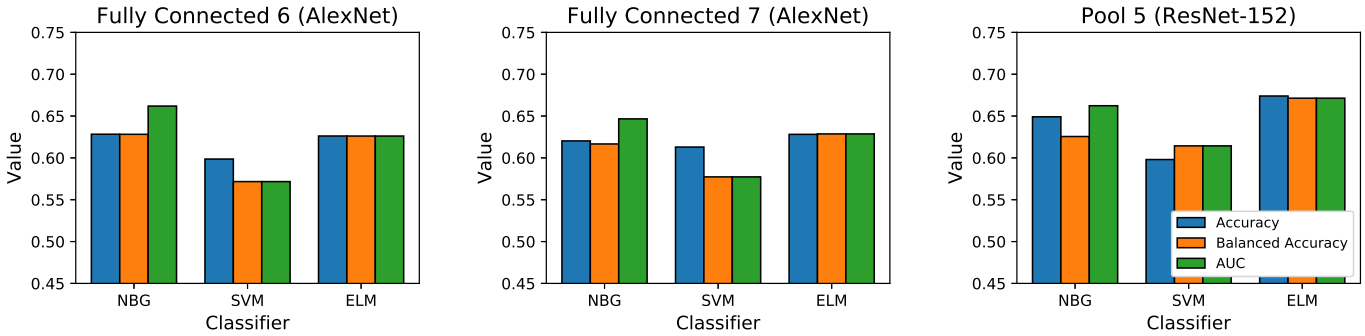


Fig. 7. Resultados de 3 clasificadores entrenados con *ConvNet features* de AlexNet y ResNet en términos de *accuracy*, *balanced accuracy* y AUC.

La reducción que vamos a ver a continuación, también tiene el objetivo de disminuir el tamaño de los vectores de características que se extraen de las redes, con el fin de poder utilizar modelos más complejos con esta técnica de Transfer Learning.

C. Reducción de las capas densas mediante finetuning

Al igual que en la extración de *ConvNet features*, para el proceso de *finetuning* es necesario disponer de modelos pre-entrenados en problemas similares para utilizar la información en nuestro beneficio.

AlexNet y VGG16 son redes convolucionales que siguen estructuras muy parecidas y que cuyas tres capas finales son del tipo *fully connected* o densas, y la última corresponde con la salida de las 1000 etiquetas del problema de ImageNet. Podemos ver la estructura de ambas redes en las Fig. 8 y 9, respectivamente. Las dos capas densas previas a la salida de la red, son llamadas "fc6" y "fc7", y tienen dimensiones de 4096 nodos en cada una.

Como se ha comentado antes, la técnica de *finetuning* consiste en realizar modificaciones a una red pre-entrenada, para adaptar la salida del modelo a nuestro problema (Fig. 2). En el caso actual, donde la evaluación de la calidad estética se realiza mediante una clasificación binaria y las redes pre-entrenadas son del problema de ImageNet, sustituimos la capa densa de salida con 1000 nodos por una de 2 nodos.

En los trabajos propuestos que hacen uso de *finetuning*, todas las capas del modelo, exceptuando la de *output*, permanecen inalterables. Sin embargo, consideramos que esta

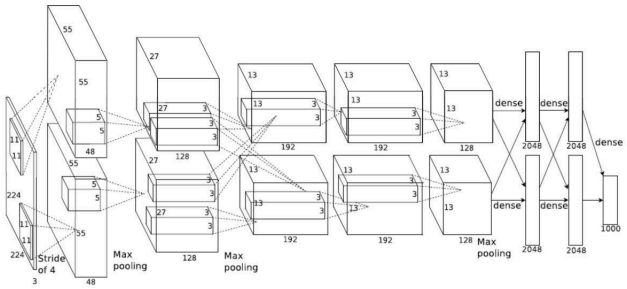


Fig. 8. Estructura original de AlexNet en dos columnas.

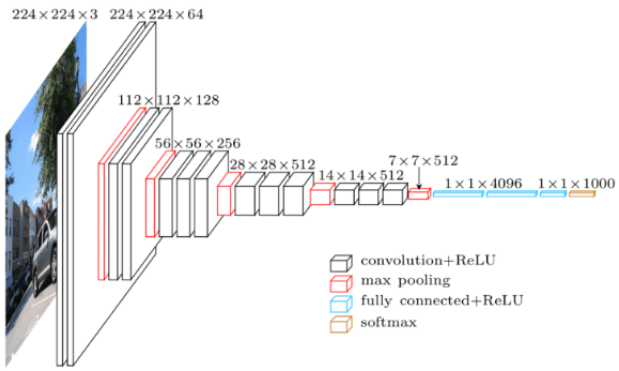


Fig. 9. Estructura de la red VGG16.

Modelo	Nodos en las capas densas	Nº de params de la red	Tamaño del modelo
AlexNet	4096	60M	223MB
	1000	14M	54MB
	500	8,6M	33MB
	250	6,1M	24MB
VGG16	4096	134M	513MB
	1000	41M	156MB
	500	28M	106MB
	250	21M	81M

TABLA I
NÚMERO DE PARÁMETROS (EN MILLONES) DE DIFERENTES REDES Y DEL ESPACIO NECESARIO EN MEMORIA (MEGABYTES) PARA ALMACENARLA, EN FUNCIÓN DEL TAMAÑO DE LAS CAPAS DENSAS.

Modelo	Nodos en las capas densas	Accuracy	Balanced Accuracy	AUC
AlexNet	4096	0.70	0.69	0.76
	1000	0.70	0.67	0.74
	500	0.69	0.67	0.73
	250	0.70	0.66	0.73
VGG16	4096	0.70	0.71	0.79
	1000	0.72	0.71	0.79
	500	0.74	0.71	0.79
	250	0.72	0.71	0.79

TABLA II
RESULTADOS EN TÉRMINOS DE ACCURACY, BALANCED ACCURACY Y AUC DE LAS REDES EN BASE AL TAMAÑO DE LAS CAPAS DENSAS.

técnica de Transfer Learning puede utilizarse de forma más eficaz aplicando modificaciones a las capas previas "fc6" y "fc7", tanto en AlexNet como en VGG16.

En este trabajo, se propone reducir el número de nodos de las tres últimas capas de los modelos AlexNet y VGG16 para resolver la clasificación binaria de la calidad estética. Las capas densas son las que concentran el mayor número de parámetros de una red, por lo que mediante este proceso, disminuiremos considerablemente el tamaño de la red.

D. Implementación

Para este trabajo se ha utilizado la librería TensorFlow [19] en Python para realizar el proceso de *finetuning* a las capas "fc6", "fc7" y el *output* de las redes neuronales AlexNet y VGG16. En ambos casos se ha utilizado el optimizador SGD (Stochastic Gradient Descent) con un learning rate de 0.001 con un tamaño de batch de 128 y se han realizado 10 epochs. Para las capas de Dropout se ha utilizado un factor de 0,5. Todo esto se ha llevado a cabo sobre una GPU Tesla K40c, donde los tiempos de entrenamiento son de 6-7 horas para AlexNet y de 23 horas para VGG16.

IV. RESULTADOS

Se han realizado experimentos reduciendo los nodos de las capas densas "fc6" y "fc7" a 1000, 500 y 250. En la tabla I se muestra la diferencia de tamaño de las redes y la memoria necesaria. Como se puede observar, la mayoría de los parámetros de nuestra red se encuentran en estas capas densas, por lo que al reducir su tamaño afecta significativamente al modelo.

Una vez reentrenadas las redes, vamos a comparar sus resultados utilizando tres métricas de evaluación sobre el conjunto de test. Estas son el *accuracy* o tasa de acierto, el *balanced accuracy* y el valor AUC (Area Under the Curve).

En la tabla II se puede observar que la modificación de las capas densas en AlexNet afecta al *balanced accuracy* y al AUC. Existe un empeoramiento de un 3% del modelo con las capas densas de tamaño 250, frente a las de 4096, pero la red reducida ocupa un 10% de memoria con respecto a la original. Con VGG16 los resultados son casi idénticos en todas las redes, independientemente del tamaño de las capas densas. Cabe destacar que sólo el *accuracy* se ha visto afectado y que la red con las capas densas de tamaño 500 funcionan mejor que las originales de 4096.

V. CONCLUSIÓN

En este trabajo se ha presentado una estrategia basada en *finetuning* capaz de reducir el tamaño de redes pre-entrenadas, en este caso AlexNet y VGG16, sin perder eficacia en la resolución del problema de la calidad estética.

Hemos visto como podemos obtener los mismos resultados con redes del 10% del tamaño de las propuestas hechas hasta ahora. Esto permitirá que los requisitos para la evaluación de nuevos casos sea mucho menor, permitiendo utilizar dichos modelos en dispositivos como móviles o, por ejemplo, una Raspberry. Estamos seguros de que es un gran paso para el desarrollo de aplicaciones reales que se beneficien del Deep Learning para la evaluación de la calidad estética.

Como trabajo futuro, planeamos seguir optimizando las redes pre-entrenadas en el problema de la calidad estética, especialmente los diseños presentados en [15], donde la salida es la distribución de los votos, en vez de la clasificación binaria en "snapshots" y "professional shots".

La reducción del número de nodos de las últimas capas de las redes neuronales, además del ahorro de memoria que supone, permite extraer *ConvNet features* con una menor dimensionalidad, pero con la misma capacidad descriptiva. Esto es de especial utilidad para aprender modelos más complejos que antes no podíamos, como por ejemplo, algunos clasificadores probabilísticos, ya que estos modelos manejan de forma natural la incertidumbre y son especialmente útiles en problemas donde la clase no está definida.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado con los fondos FEDER y el Gobierno Español (MICINN) a través del proyecto TIN2016-77902-C3-1-P.

REFERENCES

[1] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *International conference on machine learning*, 2014, pp. 647–655.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich *et al.*, "Going deeper with convolutions." Cvpr, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[6] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[8] Y. Deng, C. C. Loy, and X. Tang, "Image aesthetic assessment: An experimental survey," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 80–106, 2017.

[9] R. Datta, J. Li, and J. Z. Wang, "Algorithmic inferencing of aesthetics and emotion in natural images: An exposition," in *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*. IEEE, 2008, pp. 105–108.

[10] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Studying aesthetics in photographic images using a computational approach," in *European Conference on Computer Vision*. Springer, 2006, pp. 288–301.

[11] Y. Ke, X. Tang, and F. Jing, "The design of high-level features for photo quality assessment," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1. IEEE, 2006, pp. 419–426.

[12] L. Marchesotti, F. Perronnin, D. Larlus, and G. Csurka, "Assessing the aesthetic quality of photographs using generic image descriptors," in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1784–1791.

[13] N. Murray, L. Marchesotti, and F. Perronnin, "Ava: A large-scale database for aesthetic visual analysis," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 2408–2415.

[14] X. Lu, Z. Lin, H. Jin, J. Yang, and J. Z. Wang, "Rapid: Rating pictorial aesthetics using deep learning," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 457–466.

[15] H. Talebi and P. Milanfar, "Nima: Neural image assessment," *IEEE Transactions on Image Processing*, vol. 27, no. 8, pp. 3998–4011, 2018.

[16] F. Rubio, M. J. Flores, and J. M. Puerta, "Drawing a baseline in aesthetic quality assessment," in *Tenth International Conference on Machine Vision (ICMV 2017)*, vol. 10696. International Society for Optics and Photonics, 2018, p. 106961M.

[17] X. Tian, Z. Dong, K. Yang, and T. Mei, "Query-dependent aesthetic model with deep learning for photo quality assessment," *IEEE Transactions on Multimedia*, vol. 17, no. 11, pp. 2035–2048, 2015.

[18] Z. Dong, X. Shen, H. Li, and X. Tian, "Photo quality assessment with dcnn that understands image well," in *International Conference on Multimedia Modeling*. Springer, 2015, pp. 524–535.

[19] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

Redes Neuronales Convolucionales para Una Clasificación Precisa de Imágenes de Corales

Anabel Gómez-Ríos, Siham Tabik, Julián Luengo and Francisco Herrera
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada
Granada, España
{anabelgrios, julianlm, herrera}@decsai.ugr.es, siham@ugr.es

ASM Shihavuddin
Dpto. Matemática Aplicada y Ciencias de la Computación
Universidad Técnica de Dinamarca
Kgs. Lyngby, Dinamarca

Bartosz Krawczyk
Dpto. Ciencias de la Computación
Universidad Virginia Commonwealth
Virginia, EEUU

***Resumen*—El reconocimiento de especies de corales basado en imágenes submarinas de texturas plantea una gran dificultad para los algoritmos de aprendizaje automático, debido a la naturaleza de los datos y las características que tienen. Entre otras, encontramos que los conjuntos de datos no incluyen información sobre la estructura global del coral, que muchas especies de coral tienen características muy similares y que definir los límites espaciales entre clases es difícil ya que muchos corales tienden a vivir juntos. Por ello, la clasificación de especies de corales siempre ha requerido de la ayuda de un experto. El objetivo de este trabajo es desarrollar un modelo de clasificación precisa para imágenes de textura de corales. Nos hemos centrado en dos conjuntos de datos de imágenes de texturas y hemos analizado 1) varias arquitecturas de redes neuronales convolucionales, 2) *transfer learning* y 3) técnicas de *data augmentation*. Hemos alcanzado los porcentajes de clasificación más altos usando diferentes variaciones de ResNet en ambos conjuntos de datos.**

***Index Terms*—Redes Neuronales Convolucionales, Clasificación, Imágenes de Corales, Inception, ResNet, DenseNet**

I. Introducción

Los arrecifes de coral son ecosistemas marinos típicos de los mares cálidos y poco profundos de los trópicos. Se crean por la acumulación de esqueletos de carbonato de calcio que las especies de coral duro dejan cuando mueren. Otros corales viven después en ellos y expanden el arrecife. Son uno de los ecosistemas más valiosos del mundo al ser extremadamente biodiversos. Soportan hasta dos millones de especies y una cuarta parte de toda la vida marina en el mundo [1]. Son también muy importantes desde el punto de vista humano [2]: ayudan a limpiar el agua eliminando el nitrógeno y el carbono, son una fuente de investigación médica y económica a partir de la pesca y el turismo, una barrera natural contra huracanes y tormentas y su estudio ayuda a comprender fenómenos climáticos del pasado debido a la cantidad de años que tienen.

Este trabajo ha sido realizado con el apoyo del Ministerio de Ciencia y Tecnología de España bajo el proyecto TIN2017-89517-P y de la Junta de Andalucía bajo el proyecto P11-TIC-7765. Siham Tabik cuenta con una beca del programa Ramón y Cajal (RYC-2015-18136) y Anabel Gómez-Ríos con una beca FPU 998758-2016.

El estudio de la distribución de los arrecifes de coral a lo largo del tiempo provee información sobre el impacto del calentamiento global y los niveles de contaminación del agua. Según [1], ya se ha perdido el 19 % del área de arrecifes de coral desde la década de 1950 y, según la Unión Internacional para la Conservación de la Naturaleza (IUCN por sus siglas en inglés) y su Lista Roja de Especies Amenazadas [3], en 2017 había 237 especies amenazadas sólo en el evaluado 40 % del total estimado de especies. Esto se debe en gran medida a problemas causados por humanos: la contaminación del agua, el cambio climático, ya que los corales no toleran los cambios de temperatura, y el dióxido de carbono emitido a la atmósfera, donde un cuarto es absorbido por el océano.

Con los avances en la adquisición de imágenes y el creciente interés de la comunidad científica, se están recolectando una gran cantidad de datos sobre corales. Sin embargo, es complicado llevar un registro de todas las especies porque hay miles de ellas y la taxonomía es mutable debido a nuevos descubrimientos o por cambios en las especies conocidas a medida que se adquiere más conocimiento sobre ellas. Además, algunas especies de coral pueden parecer idénticas entre ellas para un observador humano. Por esto, se ha necesitado siempre de un experto biólogo para obtener una buena clasificación. Si conseguimos automatizar esta clasificación usando la cantidad de imágenes de corales que se están recogiendo, podemos ayudar a los científicos a estudiar más de cerca esa cantidad de datos. De hecho, la automatización de la clasificación de imágenes de corales se ha abordado en algunos trabajos. La mayoría de ellos [4], [5], [6], [7] usan modelos de aprendizaje automático combinados con técnicas de mejora de imagen y varios extractores de características. Entre estos trabajos, sólo [6] utiliza varios conjuntos de datos.

En los últimos años, las redes neuronales convolucionales (CNNs por sus siglas en inglés) han mostrado una precisión excepcional para la clasificación de imágenes [8], [9], especialmente en el campo de la visión por computador. Actualmente, sus aplicaciones se extienden a muchos campos

en los que se requiere un análisis de imágenes. El uso de CNNs en clasificación de corales supone un reto por varias causas: las diferencias entre imágenes de la misma clase, las variaciones de luz debidas al agua o el hecho de que algunas especies de coral tienden a aparecer juntas. Por otra parte, las CNNs necesitan grandes conjuntos de datos para lograr un buen rendimiento. En la práctica, se usan dos técnicas para sobrevenir esta limitación: *transfer learning*, que consiste en usar el conocimiento obtenido de otro problema, usualmente más grande, y *data augmentation*, que consiste en aumentar artificialmente el conjunto de datos de entrenamiento. Hay algunos trabajos que utilizan CNNs para clasificación de corales [10], [11], [12], pero evalúan CNNs más simples, como VGGnet o LeNet y sólo usan un conjunto de datos.

En este trabajo se propone utilizar CNNs más nuevas y con mayor capacidad para superar las limitaciones de las CNNs que se han usado anteriormente. Queremos desarrollar un sistema más preciso que se enfrente a los problemas específicos de la clasificación de imágenes de corales usando varios conjuntos de datos. En particular, hemos considerado tres de las las CNN más prometedoras, Inception v3 [13], ResNet [14] y DenseNet [15]. Inception es una nueva versión de GoogleNet [16], que ganó el premio *ImageNet Large Scale Visual Recognition Competition* (ISLVRC) [9] en 2014. ResNet ganó la misma competición en 2015 y DenseNet mejoró los resultados de ResNet en 2016. Para la clasificación, hemos considerado dos conjuntos de imágenes submarinas de corales, RSMAS y EILAT [17], que tienen la particularidad de que muestran texturas de corales, no los corales completos. Además, son conjuntos de datos pequeños que contienen muchas clases. Por otra parte, hemos comparado nuestros resultados con el modelo actual más preciso para estos dos conjuntos de datos, el propuesto por Shihavuddin y otros en [6], el cual está compuesto por distintos algoritmos clásicos de aprendizaje automático y requiere una gran supervisión.

Las contribuciones de este trabajo son las siguientes:

- Analizar las tres CNNs que hemos evaluado en este trabajo: Inception v3, ResNet y DenseNet.
- Evaluar estas tres CNNs y analizar su rendimiento usando *transfer learning* desde ImageNet.
- Analizar distintas técnicas de *data augmentation* en este tipo concreto de imágenes.
- Comparar nuestros resultados con el actual estado del arte en EILAT y RSMAS.

El resto del trabajo está organizado de la siguiente forma: en la Sección II estudiamos y analizamos Inception, ResNet y DenseNet. En la Sección III se comentan los avances que se han hecho en la clasificación de corales basada en imágenes. En la Sección IV se exponen las características de los conjuntos de datos que se han usado en este trabajo. En la Sección V se exponen y analizan los resultados que se han obtenido con los distintos modelos de CNN y tipos de *data augmentation*. Finalmente, en la Sección VI se muestran las conclusiones obtenidas y los trabajos futuros.

II. Convolutional Neural Networks

Las CNNs han logrado precisiones excepcionales en varias aplicaciones actuales [18]. De hecho, desde 2012 la prestigiosa competición ILSVRC [9] sólo la han ganado CNNs. Las capas de una CNN capturan características cada vez más complejas a medida que aumenta la profundidad de la red. En los últimos años, estas arquitecturas han ido evolucionado, primero incrementando la profundidad de las redes, después el ancho y finalmente usando características obtenidas en las primeras capas inferiores en capas más profundas. Esta sección proporciona una visión general de las CNNs utilizadas en este trabajo. Hemos considerado tres CNNs influyentes, Inception v3 (Subsección II-A), ResNet (Subsección II-B) y DenseNet (Subsección II-B). Estas tres CNNs se basan en la repetición de un módulo o bloque base compuesto por una serie de capas convolucionales, de *poolings* o de capas de normalización. La diferencia entre ellas es la composición de los módulos y cómo están distribuidos dentro de la red. Para finalizar, describimos las técnicas de optimización que hemos usado para paliar el problema de los conjuntos de datos pequeños, *transfer learning* y *data augmentation* (Subsección II-D).

II-A. Inception v3

El módulo base de Inception [13] consta de cuatro ramas en paralelo: una convolución de kernel 1×1 seguida de dos convoluciones 3×3, una convolución 1×1 seguida de una convolución 3×3, un *pooling* seguido de una convolución 1×1 y por último una convolución 1×1. La salida del módulo es la concatenación de las cuatro ramas. Inception consta en total de 10 módulos, aunque dichos módulos se van modificando ligeramente según la red se va haciendo más profunda. En concreto, se cambian cinco de los módulos con el fin de reducir el coste computacional sustituyendo las convoluciones $n \times n$ por dos convoluciones, una 1×7 seguida de una 7×1. Los dos últimos módulos sustituyen las dos últimas convoluciones 3×3 de la primera rama por dos convoluciones cada una, una 1×3 seguida de otra 3×1, esta vez en paralelo. En total, Inception v3 tiene 42 capas con parámetros.

II-B. ResNet

ResNet [14] no tiene una profundidad fija y depende del número de módulos consecutivos que se usen. Sin embargo, aumentar la profundidad de la red para obtener una mayor precisión hace que la red sea más difícil de optimizar ya que es más fácil que se produzca el problema de la desaparición de gradientes. ResNet aborda este problema ajustando una aplicación residual en lugar de la original, y añadiendo varias conexiones entre capas. Estas nuevas conexiones saltan varias capas y realizan una identidad o una convolución 1×1. El bloque base de esta red se llama *residual block* y está compuesto, cuando la red tiene 50 o más capas, por tres convoluciones secuenciales, una 1×1, una 3×3 y una 1×1, y una conexión que une la entrada de la primera convolución a la salida de la tercera convolución. En nuestro caso hemos usado dos modelos con esta arquitectura, ResNet-50, compuesta por 50 capas y ResNet-152, compuesta por 152 capas.

II-C. DenseNet

DenseNet [15] tampoco tiene una profundidad fija y depende del número de módulos que se usen. En este caso el módulo o bloque se llama *dense block* y como el de ResNet, introduce conexiones entre capas no consecutivas. En este caso, se conecta la salida de todas las capas dentro del *dense block* a la entrada de todas las capas siguientes dentro del bloque. Las conexiones entre bloques, llamadas capas de transición, funcionan como un factor de compresión en el sentido de que generan menos mapas de características de los que reciben. El bloque está compuesto de una repetición de las siguientes operaciones: *Batch Normalization* (BN), ReLU, convolución 1×1, BN, ReLU y convolución 3×3. Las capas de transición son una convolución 1×1 seguida de un *average pooling* con kernel 2×2. En este trabajo, hemos analizado DenseNet-121 y DenseNet-161, que incluyen 121 y 161 capas.

II-D. Técnicas de Optimización de CNNs

Todas las redes anteriores son demasiado profundas como para entrenarlas desde cero con nuestros conjuntos de datos, ambos muy pequeños. Por ello, hemos usado las técnicas de *transfer learning* y *data augmentation*.

La técnica de *transfer learning* consiste en usar el conocimiento aprendido de otro problema, generalmente más grande, comenzando el entrenamiento de los pesos, en lugar de con valores aleatorios, con los pesos entrenados con otro problema. En concreto, nosotros hemos usado las redes preentrenadas con ImageNet [19], y como los conjuntos de datos siguen siendo demasiado pequeños, en lugar de ajustar todos los pesos de las redes hemos añadido dos capas completamente conectadas al final de las redes anteriores. La primera de estas dos capas tiene 512 neuronas y una activación ReLU y la última tiene tantas neuronas como clases tiene el conjunto de datos que queremos clasificar y una activación softmax. De esta forma sólo entrenamos estas nuevas dos capas.

La técnica de *data augmentation* [20] consiste en aumentar artificialmente el conjunto de entrenamiento aplicando varias transformaciones a las imágenes originales, como cambiar el brillo, escalarlas, acercarlas, girarlas, reflejarlas verticalmente, etc. Las nuevas imágenes deben ser lo suficientemente reales como para que un observador externo no sepa distinguir una imagen generada por *data augmentation* de una original. En nuestro caso hemos aplicado las siguientes transformaciones:

- Desplazamiento: consiste en desplazar las imágenes horizontal y verticalmente un número determinado de píxeles calculado como una fracción del ancho o alto de la imagen. Dicha fracción se elige aleatoriamente en cada caso en el intervalo $[0, x]$ para un x dado.
- Zoom: consiste en acercar o alejar las imágenes. Dado un valor x, cada imagen será redimensionada a un valor aleatorio en el intervalo $[1 - x, 1 + x]$.
- Rotación: consiste en rotar aleatoriamente las imágenes un ángulo determinado. Dado un valor x, cada imagen se rotará un ángulo aleatorio en el intervalo $[0, x]$.
- Reflejar: consiste en reflejar las imágenes horizontalmente de forma aleatoria.

III. AVANCES EN LA CLASIFICACIÓN DE IMÁGENES DE CORALES

En esta sección comentamos los avances previos que ha habido en la clasificación automática de imágenes de corales, mostrando los retos que conlleva dicha clasificación (Subsección III-A) y los trabajos que la han analizado previamente (Subsección III-B).

III-A. Retos en la Clasificación de Imágenes de Corales

La clasificación de especies de corales basada en imágenes de texturas de corales es compleja por las siguientes razones:

- Oclusión parcial de objetos y animales.
- Variaciones en la luz debido al movimiento de mareas y a las propiedades ópticas del agua. Es común que la única fuente de luz sea la de la cámara, lo que implica iluminación no uniforme en las imágenes obtenidas.
- Anotación subjetiva de los conjuntos de datos por diferentes expertos.
- Variación en puntos de vista, distancias y calidad de imagen.
- Variabilidad en la morfología de corales que pertenecen a la misma especie.
- Separación entre clases compleja, ya que distintas especies de corales tienden a aparecer juntas.

III-B. Trabajos Relacionados

Los trabajos previos en la clasificación de imágenes de corales se pueden dividir en dos grupos: métodos que combinan algoritmos clásicos del aprendizaje automático con algoritmos de extracción de características y métodos que usan CNNs.

La mayoría de los enfoques existentes para clasificar las imágenes de corales combinan un extractor de características con un clasificador y muestran su rendimiento utilizando un único conjunto de datos [4], [5], [7]. En el primer trabajo sobre este tema [5], los autores analizaron más hábitat marino además de corales. Utilizaron un descriptor SIFT y un modelo de bolsa de características, consistente en elegir del conjunto de entrenamiento las imágenes que son más similares a cada imagen de test. En [4] se introduce el conjunto de datos *Moorea Labeled Corals* (MLC), que contiene imágenes de gran tamaño de diferentes especies de coral, y se usa SVM junto con filtros y un descriptor de textura para clasificarlo. Obtuvieron una precisión del 83,1 % sobre este conjunto de datos utilizando las imágenes de 2008 y 2009 para entrenamiento y las imágenes de 2010 para test. En [7] se utilizó un espacio de color normalizado y la transformación discreta del coseno como descriptor de textura. Sólo utilizaron un conjunto de datos, proporcionado por el *National Oceanic and Atmospheric Administration* (NOAA) del Departamento de Comercio de EE.UU.

Estos métodos no se han probado aún con nuevos conjuntos de datos. Sólo Shihavuddin y otros [6] desarrollaron un algoritmo de clasificación unificado para varios conjuntos de datos de características diferentes, entre los que podemos encontrar a RSMAS y EILAT. Los autores combinaron múltiples técnicas

de mejora de imágenes, extractores de características y clasificadores en un modelo compuesto de seis pasos. Cada paso se compone de varios algoritmos que pueden ser obligatorios u opcionales. La clasificación se realiza usando SVM, KNN, una red neuronal o la distancia media ponderada por la densidad de probabilidad. Configurando los hiperparámetros de los algoritmos y las diferentes combinaciones entre ellos, el modelo puede adaptarse a los diferentes conjuntos de datos. Este método es el estado del arte para RSMAS y EILAT. Sin embargo, implica mucha supervisión, ya que es necesario evaluar todas las combinaciones posibles de algoritmos y los hiperparámetros de cada uno de ellos.

Por otro lado, hay varios trabajos que usan CNNs. El primero de ellos fue [10]. El autor mejoró primero las imágenes de entrada mediante corrección del color y un filtrado de suavidad de la imagen para después entrenar un modelo basado en LeNet-5 en el que la capa de entrada consistía en tres canales para una imagen en color y canales adicionales para descriptores de textura y forma.

El siguiente trabajo que usó CNNs fue [11], donde los autores utilizaron VGGnet ya entrenada en ImageNet y el conjunto de datos BENTHOZ-2015 [21] para seguir entrenando la red. BENTHOZ-2015 contiene más de 400,000 imágenes y datos de sensores asociados recogidos por un vehículo autónomo sobre Australia. Los autores extrajeron varios trozos de cada imagen centrados en diferentes píxeles usando diferentes escalas. Propusieron un mecanismo para etiquetar automáticamente las imágenes de corales de forma que se obtuviera la cobertura de los corales en la región donde se recolectaron las imágenes (clasificando las nuevas imágenes como coral o no).

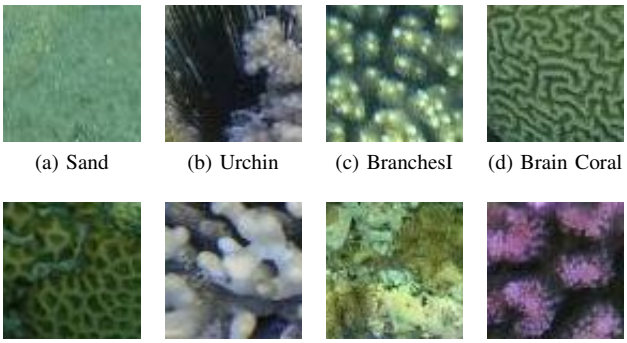
Por último, en [12] los autores utilizaron el conjunto de datos MLC y usaron CNNs junto con características extraídas de las imágenes, introduciendo un nuevo mecanismo para extraerlas. Se basaron en que las CNNs no pueden ser entrenadas desde cero usando los conjuntos de datos de corales disponibles debido a su pequeño tamaño. La extracción de características con CNNs se realizó con la red VGGnet preentrenada en ImageNet. Para clasificar ambos tipos de características, utilizaron un Perceptrón de dos capas. En sus experimentos, obtuvieron mejores precisiones con esta técnica que sólo con VGGnet, aunque la diferencia era pequeña.

Estos trabajos usan CNNs clásicas, como VGGnet o LeNet, y no usan EILAT ni RSMAS. Además, los resultados obtenidos son bajos [10], la clasificación es demasiado simple [11] o combinan las redes con otras características [12].

IV. Conjuntos de Datos

En este trabajo, hemos utilizado los conjuntos de datos RGB más recientes y pequeños que contienen el mayor número de especies de coral, RSMAS y EILAT [17]. Estos dos conjuntos de datos se componen de imágenes que muestran trozos de corales. Estos trozos capturan la textura de diferentes partes del coral y no incluyen información sobre la estructura global del mismo. Sus principales características son las siguientes:

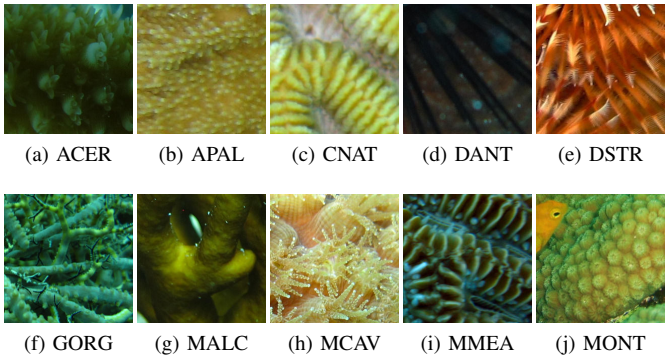
- EILAT contiene 1123 imágenes de tamaño 64×64, tomadas de arrecifes de coral cerca de Eilat en el Mar Rojo.



(a) Sand (b) Urchin (c) BranchesI (d) Brain Coral

(e) Favid Coral (f) BranchesII (g) Dead Coral (h) BranchesIII

Figura 1. Un ejemplo de cada clase de EILAT.



(a) ACER (b) APAL (c) CNAT (d) DANT (e) DSTR

(f) GORG (g) MALC (h) MCAV (i) MMEA (j) MONT

(k) PALY (l) SPO (m) SSID (n) TUNI

Figura 2. Un ejemplo de cada clase de RSMAS.

  Estas imágenes son trozos de imágenes más grandes, que fueron tomadas en iguales condiciones y con la misma cámara fotográfica. Las imágenes han sido clasificadas en ocho clases, pero las etiquetas utilizadas no corresponden a los nombres de las especies de coral. En la Fig. 1 podemos ver ejemplos de este conjunto de datos.
- RSMAS contiene 766 imágenes de tamaño 256×256. Las imágenes fueron recogidas por buzos de la Escuela Rosenstiel de Ciencias Marinas y Atmosféricas de la Universidad de Miami. Fueron tomadas bajo diferentes condiciones, con diferentes cámaras y en lugares diferentes. Las imágenes se han clasificado en 14 clases, cuyas etiquetas corresponden a los nombres en latín de la especie de coral que se muestra en cada clase. En la Fig. 2 podemos ver ejemplos de este conjunto de datos.

V. Clasificación de Imágenes de Corales usando CNNs

Esta sección está organizada en tres partes. Primero, mostramos el marco experimental que hemos usado en todos los ex-

perimentos llevados a cabo en este trabajo (Subsección V-A). Después, mostramos y analizamos los resultados de la clasificación de EILAT y RSMAS usando sólo *transfer learning* (Subsección V-B). Por último, mostramos y analizamos los resultados de las distintas técnicas de *data augmentation* en la clasificación usando el mejor modelo en cada caso hallado en la Subsección V-B (Subsección V-C).

V-A. Marco Experimental

Para evaluar todas las CNNs, hemos usado Keras [22] y Tensorflow [23] como *back-end*. Para Inception hemos usado la implementación disponible en la versión 2.0.4 de Keras y para ResNet y DenseNet, hemos adaptado el código disponible en GitHub por [24]. Todas han sido evaluadas en una gráfica NVIDIA Titan Xp.

Para la evaluación, hemos usado la medida *accuracy*, es decir, el porcentaje de acierto en la clasificación, y un esquema de validación cruzada con cinco particiones, por lo que todos los resultados que se dan en esta sección son la media de los porcentajes obtenidos sobre las cinco particiones.

Además, hemos evaluado el impacto de distintos hiperparámetros en el rendimiento de las redes, como el número de capas, el número de épocas que las entrenamos y el tamaño del *batch* que usamos para entrenarlas. En concreto, hemos usado 50 y 152 capas con ResNet, y 121 y 161 con DenseNet, con lo que tenemos cinco modelos de CNNs. Para cada uno de estos modelos, hemos probado todas las combinaciones posibles de los valores para los hiperparámetros número de épocas = $\{100, 300, 500, 700, 1000, 1300\}$ y *batch* = $\{32, 64, 128\}$.

V-B. Clasificación Usando Transfer Learning

En esta subsección analizamos el comportamiento de las redes con *transfer learning* desde ImageNet. Hemos evaluado todas las combinaciones de los hiperparámetros comentados en la subsección anterior y en la Tabla I se muestran aquellos con los que se han obtenido los mejores resultados para cada modelo junto con el *accuracy* correspondiente para dicha combinación. Se muestra además el resultado para el método de Shihavuddin, el estado del arte para estos dos conjuntos de datos. Como vemos, ResNet-152 supera al método de Shihavuddin y al resto de las CNN. Inception proporciona un mejor *accuracy* que el método de Shihavuddin en RSMAS, pero es peor en EILAT. DenseNet muestra los peores resultados en ambos conjuntos de datos. En general, estos resultados muestran que las CNNs son capaces de convertirse en el estado del arte en tareas de clasificación de corales. En RSMAS, el mejor modelo es ResNet-152, con una mejora de más del 5 % con respecto al método de Shihavuddin. En EILAT, ResNet-50 y ResNet-152 alcanzan exactamente el mismo *accuracy* y superan al método de Shihavuddin en más de un 2 %.

Los resultados obtenidos permiten concluir que sólo entrenando las últimas capas de una CNN que ya está preentrenada en ImageNet podemos superar a un método que requiere mucho tiempo y una alta supervisión humana como es el método de Shihavuddin, que se compone de seis pasos y cada paso se compone de varios algoritmos, de forma que para obtener el mejor rendimiento es necesario evaluar todas las combinaciones de algoritmos posibles a través de todos los pasos y optimizar los hiperparámetros de cada algoritmo.

V-C. Clasificación Usando Data Augmentation

En esta subsección analizamos el impacto de las técnicas de *data augmentation* enumeradas en la Subsección II-D. Tanto para EILAT como para RSMAS hemos evaluado dichas técnicas sobre los modelos que obtenían mejor rendimiento en cada caso. Para EILAT hemos usado ResNet-50 por ser más simple que ResNet-152 y obtener ambos el mismo *accuracy*. Para RSMAS hemos usado ResNet-152.

La notación `rot. = 2` significa que estamos aplicando una rotación a cada imagen de un ángulo elegido aleatoriamente para cada una en el intervalo $[0, 2]$. Esta notación es equivalente para el resto de técnicas.

La Tabla II muestra los resultados de usar *data augmentation* con ResNet-50 en EILAT y la Tabla III muestra los resultados de ResNet-152 en RSMAS. Hemos evaluado diferentes parámetros para cada técnica y varias combinaciones entre ellas, aunque en las tablas sólo se muestran las que obtuvieron un mejor resultado en cada caso. En ambos casos, la diferencia en *accuracy* entre no usar *data augmentation* y la mejor de las técnicas es muy pequeña, menos del 1 %.

La poca mejora obtenida puede explicarse por la naturaleza de las imágenes utilizadas. Dado que las imágenes originales son pequeñas y están tomadas muy de cerca, las modificaciones que aplicamos tienen que ser muy pequeñas y por tanto no tienen mucho efecto en el aprendizaje de los modelos: el desplazamiento implica perder parte de unas imágenes que ya son pequeñas y el acercamiento o zoom implica perder calidad y partes de las imágenes. Además, las imágenes están tomadas tan de cerca que la rotación y el giro no introducen variaciones significativas. Por otra parte, el rendimiento de los modelos sin usar *data augmentation* es bastante bueno, por lo que es más difícil mejorarlo.

VI. Conclusiones

La clasificación de imágenes submarinas de corales es difícil debido a la gran cantidad de especies de coral que existen, las grandes diferencias entre imágenes de una misma especie, las variaciones de luz debido al agua, o el solapamiento existente entre diferentes clases. Pocos trabajos han abordado este problema, y el único que clasifica EILAT y RSMAS es un método complejo que utiliza varios algoritmos y necesita de mucho tiempo e intervención.

Nosotros hemos abordado estos problemas utilizando algunas de las CNNs más potentes, Inception v3, ResNet y DenseNet. Hemos realizado un estudio de los fundamentos de estas CNNs, sus hiperparámetros y la posibilidad de utilizar *transfer learning* y *data augmentation*. De esta forma, hemos sido capaces de mejorar el estado del arte en EILAT y RSMAS, demostrando que las CNNs son una excelente técnica para la clasificación automática de imágenes submarinas de corales. En particular, ResNet ha sido la mejor CNN tanto en EILAT como en RSMAS.

Tabla I
Mejor accuracy, y el conjunto de hiperparámetros que lo provee en cada caso, obtenido por Inception v3, ResNet-50, ResNet-152, DenseNet-121, DenseNet-161 y el modelo de Shihavuddin. El mejor resultado está resaltado en negrita.

		Método de Shihavuddin	Inception v3	ResNet-50	ResNet-152	DenseNet-121	DenseNet-161
EILAT	*Accuracy*	95.79	96.23	**97.85**	**97.85**	91.03	93.81
	Batch	—	32	**64**	**64**	32	32
	Épocas	—	700	**500**	**300**	300	700
RSMAS	*Accuracy*	92.74	96.71	97.67	**97.95**	89.73	91.10
	Batch	—	32	64	**32**	32	64
	Épocas	—	1300	1300	**300**	700	1000

Tabla II
Mejores accuracies obtenidas por ResNet-50 en EILAT con sus mejores parámetros usando distintas técnicas de data augmentation. El mejor resultado está resaltado en negrita.

	despl. = 0.2	acer. = 0.2	rot. = 2	refl.	despl. = 0.2, acer. = 0.2
Acc.	**98.03**	97.85	97.40	97.53	97.85

Tabla III
Mejores accuracies obtenidas por ResNet-152 en RSMAS con sus mejores parámetros usando distintas técnicas de data augmentation. El mejor resultado está resaltado en negrita.

	despl. = 0.2	acer. = 0.4	rot. = 2	refl.	despl. = 0.2, acer. = 0.4
Acc.	98.36	**98.63**	97.40	97.578	98.08

Mientras que el uso de *transfer learning* sí ha dado resultados muy buenos, el uso de técnicas de *data augmentation* en este tipo de imágenes no introduce una mejora significativa en los modelos.

Este trabajo abre nuevos retos como la clasificación de especies de corales basándonos no sólo en imágenes de texturas, sino también en imágenes que contengan la estructura completa de los corales.

Referencias

[1] ESI, "Endangered species international." http://www.endangeredspeciesinternational.org/, 2017. Accessed on 13-02-2018.

[2] F. Ferrario, M. W. Beck, C. D. Storlazzi, F. Micheli, C. C. Shepard, and L. Airoldi, "The effectiveness of coral reefs for coastal hazard risk reduction and adaptation," *Nature communications*, vol. 5, p. 3794, 2014.

[3] IUCN, "Iucn red list table of number of threatened species by major groups of organisms." http://cmsdocs.s3.amazonaws.com/summarystats/2017-3_Summary_Stats_Page_Documents/2017_3_RL_Stats_Table_1.pdf, 2017. Accessed on 13-02-2018.

[4] O. Beijbom, P. J. Edmunds, D. I. Kline, B. G. Mitchell, and D. Kriegman, "Automated annotation of coral reef survey images," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 1170–1177, IEEE, 2012.

[5] O. Pizarro, P. Rigby, M. Johnson-Roberson, S. B. Williams, and J. Colquhoun, "Towards image-based marine habitat classification," in *OCEANS 2008*, pp. 1–7, IEEE, 2008.

[6] A. Shihavuddin, N. Gracias, R. Garcia, A. C. Gleason, and B. Gintert, "Image-based coral reef classification and thematic mapping," *Remote Sensing*, vol. 5, no. 4, pp. 1809–1841, 2013.

[7] M. D. Stokes and G. B. Deane, "Automated processing of coral reef benthic images," *Limnol. Oceanogr.: Methods*, vol. 7, no. 157, pp. 157–168, 2009.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[9] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[10] M. Elawady, "Sparse coral classification using deep convolutional neural networks," *arXiv preprint arXiv:1511.09067*, 2015.

[11] A. Mahmood, M. Bennamoun, S. An, F. Sohel, F. Boussaid, R. Hovey, G. Kendrick, and R. Fisher, "Automatic annotation of coral reefs using deep learning," in *OCEANS 2016 MTS/IEEE Monterey*, pp. 1–5, IEEE, 2016.

[12] A. Mahmood, M. Bennamoun, S. An, F. Sohel, F. Boussaid, R. Hovey, G. Kendrick, and R. Fisher, "Coral classification with hybrid feature representations," in *Image Processing (ICIP), 2016 IEEE International Conference on*, pp. 519–523, IEEE, 2016.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[15] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, vol. 1, p. 3, 2017.

[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[17] A. Shihavuddin, "Coral reef dataset, v2.." Mendeley data https://data.mendeley.com/datasets/86y667257h/2, 2017. Accessed on 12-02-2018.

[18] M. D. Ferreira, D. C. Corrêa, L. G. Nonato, and R. F. de Mello, "Designing architectures of convolutional neural networks to solve practical problems," *Expert Systems with Applications*, vol. 94, pp. 205 – 217, 2018.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition (CVPR), 2009. IEEE Conference on*, pp. 248–255, IEEE, 2009.

[20] S. Tabik, D. Peralta, A. Herrera-Poyatos, and F. Herrera, "A snapshot of image pre-processing for convolutional neural networks: case study of mnist," *Int J Comput Intell Syst*, vol. 10, pp. 555–568, 2017.

[21] M. Bewley, A. Friedman, R. Ferrari, N. Hill, R. Hovey, N. Barrett, E. M. Marzinelli, O. Pizarro, W. Figueira, L. Meyer, *et al.*, "Australian sea-floor survey data, with images and expert annotations," *Scientific data*, vol. 2, p. 150057, 2015.

[22] F. Chollet *et al.*, "Keras." https://github.com/keras-team/keras, 2015.

[23] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[24] F. Yu, "Resnet and densenet cnns in keras." https://github.com/flyyufelix/cnn_finetune, 2017.

I Workshop en Deep Learning (DeepL)

SESIÓN 2

Representaciones basadas en redes neuronales para tareas de recomendación

Pablo Pérez-Núñez
Centro de Inteligencia Artificial
Universidad de Oviedo en Gijón
Gijón, España
UO232368@uniovi.es

Oscar Luaces
Centro de Inteligencia Artificial
Universidad de Oviedo en Gijón
Gijón, España
oluaces@uniovi.es

Antonio Bahamonde
Centro de Inteligencia Artificial
Universidad de Oviedo en Gijón
Gijón, España
abahamonde@uniovi.es

Jorge Díez
Centro de Inteligencia Artificial
Universidad de Oviedo en Gijón
Gijón, España
jdiez@uniovi.es

***Resumen*—Los sistemas de recomendación tratan de conocer, de alguna manera, los gustos de los usuarios con el propósito de recomendar productos que sean de su agrado. La manera de representar a los usuarios tiene un rol importante en estos sistemas, ya que se espera que una buena representación facilite la correcta identificación del perfil de consumo de cada usuario, sintetizando, de alguna manera, sus gustos. No es menos importante la manera en la que se representan los productos, donde el contexto u orden en que se consumen podría ser relevante en su representación. En este artículo se analizan varias formas de representación basadas en redes neuronales y se aplican a dos tareas de recomendación diferentes en un conjunto de reproducciones musicales. Los resultados muestran que parece relevante considerar el orden de consumo para la codificación de los productos pero no para la codificación del perfil de los usuarios.**

***Index Terms*—perfiles de usuario, sistemas de recomendación, factorización, filtros colaborativos**

I. Introducción

La mayor parte de los recursos digitales que utilizamos hoy en día realizan recomendaciones a sus usuarios utilizando Sistemas de Recomendación ([1], [2]). Así sucede, por ejemplo, cuando entramos en la web de un periódico digital, donde el orden en el que están colocadas las noticias ya es, de alguna manera, una recomendación de lectura. Las listas ordenadas de noticias que aparecen en los márgenes (*"las noticias más leídas"* o *"las más comentadas"*) constituyen también una recomendación, basada en este caso, en la interacción de otros usuarios con la publicación digital. Otro ejemplo de recomendación podemos encontrarlo dentro de los artículos, en forma de enlaces, que nos conducen a otras noticias u otras recomendaciones del tipo *"noticias relacionadas"*.

Los sistemas de recomendación han encontrado en las tiendas online un campo de aplicación muy importante, con el objetivo de incrementar las ventas al tiempo que aumentar la satisfacción de los clientes. En este caso, lo que se recomienda son productos y las recomendaciones que se realizan son del estilo *"los clientes que vieron este producto también vieron..."*, *"comprados juntos habitualmente"* o incluso rankings de productos en función del número de ventas o de las críticas de los compradores. Podemos también encontrar recomendaciones en las plataformas de streaming de contenido multimedia, por ejemplo, Netflix o Spotify, en sitios web dedicados a organizar las reseñas de hoteles y restaurantes, como TripAdvisor, etc.

Algunos de los ejemplos antes mencionados son sistemas de recomendación elementales, no personalizados, que basan sus recomendaciones simplemente en comportamientos o gustos mayoritarios: si mucha gente ve una película y, además, las valoraciones que tiene ésta son muy buenas, entonces es bastante probable que se recomiende a los usuarios que todavía no la hayan visto.

Sin embargo, hay otros sistemas que basan sus recomendaciones teniendo en cuenta información adicional presente en el contexto. Uno de los más básicos ya se comentó en un párrafo anterior: cuando se detecta que un usuario está viendo información de un producto, el sistema le muestra productos que se parecen o que se compran junto con el que está viendo. Este sistema tiene en cuenta el producto que se está viendo para hacer una recomendación personalizada.

Hay otro tipo de recomendadores que almacenan un perfil para cada usuario en el que, de alguna manera, están contenidos sus gustos o preferencias [3]. Posteriormente, estos perfiles podrán combinarse para hacer recomendaciones personalizadas, con lo que se conseguirá una mayor satisfacción del usuario ante la recomendación. La forma en la que se calcule ese perfil podrá ser más o menos elaborada.

Una manera de tener en cuenta ese perfil de consumo consiste en representar cada usuario con un vector que codifique

El trabajo realizado en este artículo ha sido financiado, en parte, por el proyecto TIN2015-65069-C2-2-R del Ministerio de Economía y Competitividad y por los fondos FEDER.

la información relativa a su interacción[1] con los productos. Esto puede lograrse mediante, por ejemplo, la construcción de *embeddings* utilizando factorización de matrices [4]. De igual forma, los productos también pueden representarse por vectores que resuman la interacción que se ha tenido con ellos. En este artículo vamos a explorar los beneficios que pueden obtenerse al utilizar dos técnicas basadas en redes neuronales, *word2vec* [5] y *doc2vec* [6], para la codificación de usuarios y de productos. Estos algoritmos han sido diseñados originalmente para la codificación de palabras y documentos en procesos que requieren representar textos para tareas de aprendizaje. La codificación que obtienen está basada en el contexto de cada palabra en cada documento. El consumo de cierto tipo de productos también guarda relación con el contexto temporal, por ejemplo, la reproducción de una secuencia de canciones (*playlist*), por lo que podemos utilizar estas técnicas para las tareas de codificación. Con objeto de mostrar la relevancia de este orden secuencial, en este trabajo comparamos la codificación obtenida mediante estas técnicas frente a otra, que se obtiene al construir un embedding a partir de una codificación *one-hot*, donde no se tiene en cuenta el orden de consumo en manera alguna.

Una decisión que debe ser adoptada para codificar los perfiles de consumo de los usuarios es cuánto nos vamos a remontar en la secuencia temporal para construir y codificar dicho perfil. En este artículo hemos considerado y comparado la codificación que se obtiene considerando una trayectoria de consumo reciente y una trayectoria de consumo a largo plazo. Es posible que, en determinadas situaciones, sea interesante mantener estos dos perfiles, sobre todo cuando los productos para los que se quiere realizar recomendaciones tengan un comportamiento estacional. Ejemplos de este tipo pueden encontrarse en la alimentación, donde hay algunos platos que se consumen más durante el invierno y otros durante el verano.

Los sistemas de recomendación pueden clasificarse en dos grandes grupos: por una parte podemos construir *filtros colaborativos* [7], en los que se utiliza únicamente la información que se tiene sobre la interacción de los usuarios con los productos y este comportamiento puede darnos una idea de sus gustos o preferencias. Por otra parte, existen también sistemas de recomendación *basados en contenido* [8], donde se utiliza información conocida de los usuarios, como puede ser su ubicación, el sexo, etc. e información conocida de los productos, por ejemplo, género y fecha de estreno si se tratase de películas. En este tipo de sistemas se pueden tener perfiles de clientes y productos basándose en esos datos conocidos.

En este artículo hemos utilizado únicamente filtros colaborativos puesto que queremos estudiar la utilidad de codificar perfiles que resuman la interacción registrada entre usuarios y productos. Incluir información basada en contenido podría distorsionar este estudio, si bien es probable que los resultados en cuanto a precisión en la recomendación puedan ser mejores, pues dispondríamos de más información.

La organización del artículo se muestra a continuación: en la siguiente sección se hace una breve reseña de trabajos previos relacionados con la elaboración de perfiles. La Sección III se dedica a detallar cómo se pueden codificar los perfiles utilizando las técnicas incluidas en word2vec y doc2vec, que se utilizarán para codificar y comparar dos tipos de perfiles, uno a largo plazo (*perfil consolidado*) y otro a corto plazo (*perfil reciente*) para cada usuario. Seguidamente, en la Sección IV, se detallarán dos tareas de recomendación en las que se va a comprobar el rendimiento de estos perfiles. En la Sección V, dedicada a los resultados, se mostrará el rendimiento de los perfiles en las dos tareas planteadas, discutiendo los resultados obtenidos. Finalmente, se presentan las conclusiones y se apunta alguna línea futura de investigación en este contexto.

II. Trabajo relacionado

Son varios los trabajos que abordan la creación de perfiles utilizando la factorización de matrices como herramienta.

En [9] los autores presentan un algoritmo que proyecta cantantes y canciones en un nuevo espacio utilizando factorización de matrices. Las tareas de aprendizaje que se abordan en este trabajo están relacionadas con la predicción de artistas que han podido interpretar una canción, la predicción de canciones que han podido ser interpretadas por un determinado artista, la obtención de canciones similares (en algún sentido) a una dada o la obtención de artistas similares a uno dado. Estas preguntas son, realmente, tareas de recomendación y las proyecciones de cantantes y canciones pueden ser consideradas perfiles.

Algo similar se plantea en [10], donde en este caso se calculan los representaciones de usuarios y canciones con el objetivo de generar *playlists* del agrado de los usuarios.

La factorización de matrices también se ha utilizado para condensar los gustos de las personas respecto a productos alimenticios. En [4] los autores obtuvieron perfiles de consumidores útiles para conocer sus preferencias respecto al consumo de carne con diferentes periodos maduración.

Hasta donde nosotros sabemos, no hay trabajos en los que se plantee una codificación de los productos basándose en el contexto u orden en el que los usuarios interactúan con ellos. Tampoco tenemos constancia de trabajos en los que se discuta la necesidad de mantener dos perfiles para cada usuario, uno representando los gustos consolidados del usuario y otro representando los recientes.

III. Creación de perfiles a partir de la interacción de los usuarios con los productos

Supongamos un conjunto de usuarios, $\mathcal{U}$, y un conjunto de productos, $\mathcal{P}$. Además, conocemos las interacciones que ha tenido cada usuario con los productos y en qué orden se ha producido dicha interacción, de tal manera que para cada usuario disponemos de una lista ordenada de productos con los que ha interactuado a lo largo del tiempo:

$$\mathcal{D} = \{(\boldsymbol{u}, \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots) : \boldsymbol{u} \in \mathcal{U}, \boldsymbol{p}_i \in \mathcal{P}\} \quad (1)$$

[1] A lo largo de este artículo utilizamos la palabra *interacción* para referirnos a cualquier forma de consumo de un producto: la compra de un item, escuchar una canción, ver un producto en una tienda on-line, leer una noticia en una publicación digital, etc.

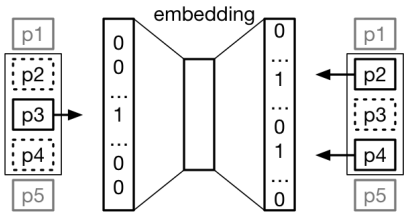


Figura 1. Arquitectura del word2vec utilizando skip-gram con una ventana de tamaño 1 para definir el contexto, que será la suma de los vectores one-hot de las palabras contenidas en el mismo

A partir de esta información aplicaremos los algoritmos word2vec y doc2vec para obtener los perfiles de productos y usuarios.

Los conjuntos $\mathcal{U}$ y $\mathcal{P}$ contienen únicamente los identificadores de los usuarios y productos, respectivamente. Una representación habitual para los elementos de estos conjuntos consiste en utilizar vectores *one-hot*, que son vectores cuya dimensión es igual a la cardinalidad del conjunto de elementos que representan. Un vector one-hot que represente al i-ésimo elemento del conjunto tiene un 1 en la componente i y todas las demás son 0. Los algoritmos word2vec y doc2vec, que describimos brevemente a continuación, se encargarán de recodificar estos vectores, calculando unos *encajes* (*embeddings*) en los que se tiene en cuenta la secuencia de aparición de los distintos elementos en los datos de entrada.

III-A. Codificación de los productos

Word2vec [5] es un algoritmo ideado para codificar las palabras de un corpus mediante vectores, de tal manera que éstos se disponen en el espacio de acuerdo a cómo se relacionan en los textos del corpus. Así, aquellas palabras con representaciones próximas suelen tener una relación fuerte.

Los vectores aprendidos para las palabras codifican ciertos patrones lingüísticos, que quedan patentes al realizar operaciones con los vectores resultantes, ya que, como se explica en [5], si al vector que representa la palabra *Madrid* se le resta el que representa la palabra *España* y se le suma el de *Francia*, resultará un vector muy cercano al que representa *París*.

Los autores plantean dos posibles estrategias para la recodificación de palabras, que resultan en dos tareas de aprendizaje:

- *Continuous Bag of Words (CBOW)*, donde a partir de las palabras del contexto se trata de predecir la central, y
- *Skip-gram*, donde a partir de la palabra central se trata de predecir las del contexto (Figura 1).

En ambos casos, el contexto se define mediante un tamaño de ventana y se representa como un vector que es la suma de los vectores one-hot de las palabras contenidas en él. La Figura 1 muestra un esquema de word2vec utilizando skip-gram, en el instante en que se presenta como entrada la palabra $\boldsymbol{p}_3$ y como salida deseada su contexto que, en este caso, son las palabras $\boldsymbol{p}_2$ y $\boldsymbol{p}_4$ o, más específicamente, el vector suma de los vectores que representan a dichas palabras. Entre la entrada y la salida hay una capa oculta, en la que obtendremos la representación buscada tras el proceso de entrenamiento.

Los autores afirman en su artículo que la estrategia skip-gram ofrece mejores resultados y, por tanto, esta estrategia es la que utilizaremos para la obtención de las representaciones de los productos en nuestro sistema de recomendación.

En un símil con el tratamiento de textos, nosotros consideramos la lista de interacciones con productos de la Ecuación (1) como si fuese una secuencia de palabras en un texto de forma que, para un determinado tamaño de ventana trataremos de aprender a predecir con qué productos ha habido interacción antes y después de la interacción con uno determinado.

III-B. Codificación de los usuarios

Tras aprender la codificación de los productos a partir de cómo los usuarios han interactuado con ellos, podemos obtener el perfil de cada usuario. Para ello proponemos utilizar el algoritmo doc2vec [6], que ha sido diseñado para codificar documentos o párrafos a partir de palabras codificadas mediante word2vec. En su artículo, los autores afirman que las representaciones obtenidas por este algoritmo mejoran el rendimiento en tareas de Recuperación de Información respecto al popular *Bag Of Words*. Para aprender la codificación de documentos, los autores también sugieren dos arquitecturas:

- *Distributed Memory version of Paragraph Vector (PV-DM)*, en la que se aprende a codificar cada documento en un proceso en el que se trata de predecir cuál es la siguiente palabra a una secuencia (ventana) de palabras del documento. La entrada al sistema es la concatenación del vector que representa al documento con los que representan a las palabras de la ventana seleccionada. El aprendizaje únicamente modifica la codificación de cada documento, minimizando el error de predicción, mientras que la codificación de las palabras (obtenida previamente mediante word2vec) se mantiene fija.
- *Distributed Bag of Words version of Paragraph Vector (PV-DBOW)*, donde se toma un documento y se eligen al azar unas cuantas palabras del mismo. A partir únicamente del vector que representa el documento se aprende su codificación para predecir la presencia, sin importar el orden, de las palabras anteriormente elegidas.

El objetivo final en ambos casos es ir modificando durante el entrenamiento la representación de cada documento. En [6], los autores muestran el buen rendimiento de estas representaciones, proponiendo el cálculo de ambas, PV-DM y PV-DBOW, para su utilización de manera concatenada.

En nuestro trabajo consideraremos la lista de productos con los que un usuario ha interaccionado, Ecuación (1), como el equivalente a un documento, de manera que habrá un documento por cada usuario. Podremos entonces utilizar doc2vec para la obtención de un vector que codifique a cada usuario. En otras palabras, identificaremos el perfil de un usuario por la secuencia de productos con los que tuvo interacción.

III-C. Perfil consolidado y perfil reciente

Una vez que se han presentado las herramientas con las que trabajaremos, vamos a definir cómo se pueden obtener el perfil consolidado y el perfil reciente de los usuarios.

Esencialmente, el procedimiento es el mismo para ambos perfiles, pero la diferencia radica en los datos con que entrenaremos la red doc2vec. Así, para obtener un perfil consolidado utilizaremos toda la información disponible acerca de las interacciones del usuario con los productos, independientemente del momento en que se haya producido cada interacción. Nuestra intención es que el perfil consolidado registre o codifique todos los intereses del usuario a lo largo del tiempo.

Sin embargo, para efectuar cierto tipo de recomendaciones puede ser de poca utilidad la información relativa al consumo mucho tiempo atrás. Esto sucede, por ejemplo, en la recomendación de listas de reproducción de canciones, en las que, para añadir un nuevo tema del agrado del usuario parece obvio que es más importante tener en consideración las canciones que acaba de escuchar, que aquellas que escuchó hace semanas, o meses. Por esta razón hemos considerado la codificación de perfiles recientes, que se obtendrán entrenando doc2vec únicamente con las interacciones más recientes del usuario. Obviamente, el umbral de decisión que determina qué interacciones son recientes y cuáles no lo son dependerá del tipo de productos que se vayan a recomendar.

IV. Tareas de recomendación

Vamos a estudiar el rendimiento de los mecanismos de representación de productos y usuarios antes mencionados en dos problemas que se detallan a continuación. Los resultados (Sección V) se compararán con los obtenidos utilizando la codificación basada en vectores one-hot.

IV-A. Tarea 1: ¿interactuará con un determinado producto?

Para resolver esta tarea contamos con un conjunto de entrenamiento cuyos ejemplos son tripletas que contienen un usuario, un producto y si dicho usuario ha interactuado o no con dicho producto, esto es,

$$\mathcal{D}_1 = \{(\boldsymbol{u}, \boldsymbol{p}, z) : \boldsymbol{u} \in \mathcal{U}, \boldsymbol{p} \in \mathcal{P}, z \in \{+1, -1\}\}. \quad (2)$$

El algoritmo que diseñamos para resolver esta tarea aprenderá utilizando la siguiente función logística:

$$\Pr(z|\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{W}, \boldsymbol{V}) = \sigma(z \cdot g(\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{W}, \boldsymbol{V})), \quad \text{con } \sigma(x) = \frac{1}{1+e^{-x}}, \quad (3)$$

donde $\boldsymbol{W}$ y $\boldsymbol{V}$ son parámetros que deben ser encontrados utilizando la estimación *Máxima Probabilidad a Posteriori (MAP)*, y g es una función de compatibilidad entre los usuarios y los productos, definida como el producto escalar:

$$\mathrm{g}(\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{W}, \boldsymbol{V}) = \langle \boldsymbol{W}\boldsymbol{u}, \boldsymbol{V}\boldsymbol{p} \rangle. \quad (4)$$

Para enriquecer la expresividad del modelo aprendido se incorporarán términos independientes.

La función de pérdida a minimizar es, en este caso,

$$-\log \prod_{(\boldsymbol{u},\boldsymbol{p},z)} \Pr(z|\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{W}, \boldsymbol{V})) = \log\left(1 + e^{-z\langle \boldsymbol{W}\boldsymbol{u}, \boldsymbol{V}\boldsymbol{p}\rangle}\right), \quad (5)$$

también conocida como *softplus*.

IV-B. Tarea 2: ¿cuál será el siguiente producto?

La segunda de las tareas a resolver es tratar de anticiparnos a la interacción inmediatamente siguiente del usuario. En este caso el conjunto de entrenamiento estará formado por tripletas conteniendo el usuario, $\boldsymbol{u}$, el último producto con el que ha interactuado, $\boldsymbol{p}_i$ y el siguiente producto con el que interactúa a continuación, $\boldsymbol{p}_j$,

$$\mathcal{D}_2 = \{(\boldsymbol{u}, \boldsymbol{p}_i, \boldsymbol{p}_j) : \boldsymbol{u} \in \mathcal{U}, \boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{P}\}. \quad (6)$$

En este caso el algoritmo podría diseñarse aprendiendo la siguiente función:

$$\Pr(y = j|\boldsymbol{u}, \boldsymbol{p}_i, \theta) = \frac{e^{v_j}}{\sum_k e^{v_k}} = \mathrm{softmax}(v)_j \quad (7)$$

siendo j el identificador del producto $\boldsymbol{p}_j$, k el de cualquier producto, θ el conjunto de parámetros (pesos de una red neuronal, por ejemplo) que se deben aprender y v el valor de salida de la red cuya entrada es la concatenación del usuario y el producto. También incorporamos un término independiente.

La función de pérdida habitual en problemas de este tipo es la *entropía cruzada* (*cross entropy*). Sin embargo, cuando el número de clases es elevado, algo común en los problemas de recomendación, el cálculo de $\mathrm{softmax}(v)$ es muy costoso, por lo que se recurre a estrategias de estimación del error, como la denominada *noise-contrastive estimation (NCE)* [11], que hemos utilizado en este trabajo.

V. Resultados experimentales

En esta sección describimos y analizamos los resultados experimentales que hemos obtenido utilizando distintas codificaciones para usuarios y productos. Más concretamente, hemos analizando el rendimiento de tres perfiles diferentes para los usuarios y dos para los productos.

V-A. Descripción del conjunto de datos

Para la experimentación hemos utilizado un conjunto de datos de la web *www.last.fm*, que han sido previamente publicados en el capítulo 3 del libro [12] y que se encuentran disponibles públicamente[2]. El conjunto contiene las reproducciones de música de 992 usuarios durante 5 años, aproximadamente. En total hay algo más de 19 millones de reproducciones sobre un conjunto de un millón y medio de canciones. En el conjunto también aparece la fecha y hora exactas de cada reproducción, con lo que no es complicado elaborar para cada usuario una lista ordenada como la que se muestra en la Ecuación (1).

V-B. Preparación de los experimentos

Hemos filtrado el conjunto, eliminando canciones repetidas o que se habían escuchado muy poco, quedándonos sólo con aquellas que se han escuchado al menos 10 veces, con lo que hemos reducido el conjunto a 312895 canciones.

Cada lista de reproducción asociada a un usuario fue separada en una parte para entrenar los perfiles (75 %) y otra para utilizar en las tareas propuestas (25 %), ver Figura 2.

[2] http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/lastfm-1K.html

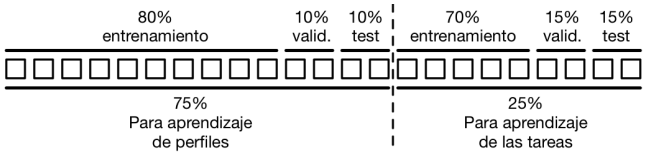


Figura 2. Método que se utiliza para la separación de la lista de reproducción de cada usuario en conjuntos de entrenamiento, validación y test

A su vez, cada una de estas dos partes fue separada en entrenamiento/validación/test, utilizando las partes de entrenamiento y validación para buscar los hiper-parámetros con mejor rendimiento. Los resultados que se muestran en las tablas son los obtenidos con estos modelos al evaluar su rendimiento sobre el conjunto de test.

Todos los algoritmos utilizados en este artículo fueron implementados utilizando la librería TensorFlow [13] sobre Python, y el optimizador SGD-Adam [14].

V-C. *Obtención de perfiles*

Para codificar las canciones utilizamos word2vec sobre el conjunto de datos correspondiente, usando un tamaño de ventana de 2 y con objeto de obtener vectores en un espacio de 64 dimensiones. Estos parámetros eran los que mejores resultados mostraban sobre el conjunto de validación.

Una vez codificadas las canciones, se codificaron los usuarios mediante la red doc2vec de dos maneras diferentes, una para obtener el perfil consolidado (P_{con}) y otra para el perfil reciente (P_{rec}). En ambos casos también utilizamos un espacio de 64 dimensiones, con una ventana de tamaño 2 (para la arquitectura PV-DM) y seleccionando todas las reproducciones del entrenamiento (para la arquitectura PV-DBOW). Al calcularse mediante los dos métodos propuestos en la Sección III-B y concatenarlos, finalmente los usuarios quedan proyectados en un espacio de 128 dimensiones. Para el cálculo del P_{con} se utilizaron todas las reproducciones reservadas para el aprendizaje de los perfiles, mientras que para el P_{rec} se utilizaron solo las reproducciones del último mes para cada usuario.

Finalmente, obtuvimos una codificación en la que los vectores de los usuarios y productos, en formato one-hot, son proyectados en un espacio de 64 dimensiones para las canciones y de 128 dimensiones para los usuarios. Utilizamos las mismas dimensiones que las de los perfiles obtenidos con word2vec y doc2vec con el fin de que los espacios tengan la misma capacidad expresiva y la comparación sea justa. La proyección se realiza a través de un encaje (embedding) que resulta del proceso de aprendizaje de cada tarea de recomendación, mediante el cálculo de dos matrices, X e Y, que proyectan cada usuario y cada canción en un espacio con las dimensiones antes especificadas (ver Figuras 3 y 4).

V-D. *Tarea 1*

Para la tarea de aprender un modelo capaz de indicarnos si un usuario va a escuchar una canción en el futuro, resolvemos el problema de optimización planteado en la Sección IV-A.

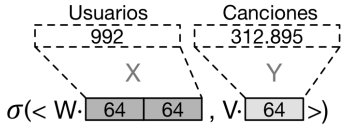


Figura 3. Red diseñada para la tarea 1, Ecuación (3). El vector que define el perfil del usuario (gris oscuro) y el de la canción (gris claro) pueden ser precalculados mediante doc2vec y word2vec, respectivamente o pueden aprenderse ad-hoc para resolver la tarea, a partir de la representación one-hot y optimizando los parámetros ($\boldsymbol{X}$ e $\boldsymbol{Y}$) necesarios.

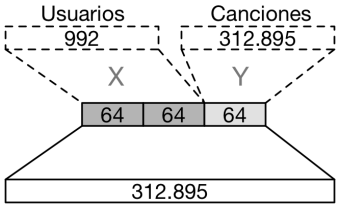


Figura 4. Red diseñada para la tarea 2, Ecuación (7). Las distintas codificaciones para usuarios y canciones son las mismas que para la tarea 1.

El conjunto $\mathcal{D}_1$ de la Ecuación (2) se construye utilizando la parte de entrenamiento de las reproducciones reservadas para el aprendizaje de las tareas (25 %), como se representa en la Figura 2. Las canciones contenidas en ese subconjunto de datos se etiquetan con $+1$, y, por cada una de ellas, se elegirá al azar una canción que el usuario no haya escuchado previamente, que se etiquetará con -1.

En la Tabla I se recogen los resultados, medidos en *Precision*, *Exhaustividad* (*Recall*) y *F1* (media armónica de las anteriores). En estos resultados se aprecia que la utilización del perfil de las canciones obtenido mediante word2vec favorece el aprendizaje. En cuanto a los perfiles de usuario, no parece que la incorporación de perfiles obtenidos mediante doc2vec mejoren el rendimiento, si bien parece más útil el perfil consolidado que el perfil reciente para este problema.

V-E. *Tarea 2*

Para resolver el problema de optimización de la Sección IV-B (predecir la siguiente canción a reproducir) utilizaremos una red con la estructura de la Figura 4, que entrenaremos con el conjunto $\mathcal{D}_2$ de la Ecuación (6), en el que los pares consecutivos de canciones reproducidas se construyen utilizando los datos reservados para el aprendizaje de tareas (25 %). El resto de datos se utiliza para obtener las codificaciones con word2vec y doc2vec, como en la tarea anterior.

Tabla I
Tabla de resultados para la tarea 1

Representación				
Usuario	Canción	Precision	Recall	F1
one-hot	one-hot	77.5	73.8	75.6
one-hot	word2vec	**83.1**	**80.2**	**81.6**
P_{con}	word2vec	80.5	79.1	79.8
P_{rec}	word2vec	77.5	74.5	76.0

Tabla II
Tabla de resultados para la tarea 2

Representación		Precisión						
Usuario	Canción	$P@5$	$P@10$	$P@20$	$P@50$	$P@100$	$P@1000$	Mediana
one-hot	one-hot	1.4	1.8	2.5	4.1	6.2	18.3	17476
one-hot	word2vec	7.7	**12.3**	**17.4**	**25.2**	**31.8**	**58.6**	**511**
P_{con}	word2vec	7.3	9.6	12.8	18.2	23.8	52.1	857
P_{rec}	word2vec	**8.3**	10.9	14.1	19.4	24.7	50.2	986

Para evaluar el rendimiento de las diferentes combinaciones de perfiles, se utilizó la medida *precision at x* ($P@x$), donde x varía en valores entre 5 y 1000. De esta manera podremos ver el porcentaje de veces que la canción que realmente va a escuchar el usuario se encuentra entre las x que más probabilidad les otorga el modelo aprendido. Los resultados de la Tabla II muestran que, nuevamente, la utilización de un perfil precalculado con word2vec para las canciones mejora el rendimiento considerablemente.

Para los perfiles de usuario, los mejores resultados se obtienen con la codificación resultante de la representación one-hot, calculada durante el aprendizaje de la tarea. Si enfrentamos el perfil reciente al consolidado, parece que el reciente ofrece mejor rendimiento, lo cual tiene sentido debido a la naturaleza de la tarea. Cabe destacar que los bajos valores de $P@5$ (por debajo del $10\,\%$ en todos los casos) se deben a que la tarea de recomendación es muy difícil: con tan sólo 5 recomendaciones se le pide al sistema acertar la siguiente canción entre 312895 alternativas posibles. La última columna de la tabla muestra la mediana de la posición que ocupa la canción que debería predecirse para acertar ($\boldsymbol{p}_j$ en la Ecuación 6) en el ranking que se obtiene atendiendo a la probabilidad predicha para cada canción. El valor 511 no es un mal resultado, teniendo en cuenta que hay más de trescientas mil canciones posibles.

Doc2vec, en su versión PV-DM, es entrenado para intentar predecir la siguiente canción, como en esta tarea 2. Los resultados que obtiene son ligeramente mejores, pero utiliza las dos últimas canciones escuchadas por el usuario.

VI. Conclusiones y trabajo futuro

Los Sistemas de Recomendación tienen muchas aplicaciones en la actualidad. Su éxito radica, principalmente, en la personalización que se está consiguiendo en las recomendaciones. Es por esto que la creación de perfiles de productos y usuarios cobra especial importancia.

En este artículo hemos evaluado el rendimiento de perfiles precalculados mediante el uso de las redes neuronales word2vec y doc2vec, y los hemos comparado con la codificación que se obtiene al calcular un embedding de los vectores de entrada a un espacio de forma que se optimiza un determinado objetivo. La comparación se ha efectuado en el contexto de dos tareas de aprendizaje: una de carácter más inmediato, predecir la siguiente canción a escuchar, y otra a más largo plazo, predecir si una canción se va a escuchar en el futuro.

Los resultados muestran que el rendimiento de los sistemas de recomendación mejora sustancialmente al introducir el perfil precalculado para las canciones mediante word2vec. Sin embargo, el perfil obtenido para los usuarios con doc2vec no mejora el rendimiento que se obtiene con la codificación obtenida a partir de vectores one-hot.

Como trabajo futuro, sería interesante obtener perfiles de usuario utilizando una red LSTM [15], ya que este tipo de redes es capaz de olvidar productos vistos hace tiempo.

Agradecimientos

Agradecemos a NVIDIA Corporation la donación de la GPU Titan Xp utilizada en esta investigación.

Referencias

[1] P. Resnick and H. R. Varian, "Recommender systems," *Commun. ACM*, vol. 40, pp. 56–58, Mar. 1997.

[2] F. Ricci, L. Rokach, and B. Shapira, "Introduction to recommender systems handbook," in *Recommender systems handbook*, pp. 1–35, Springer, 2011.

[3] J. Díez, D. Martínez-Rego, A. Alonso-Betanzos, O. Luaces, and A. Bahamonde, "Metrical representation of readers and articles in a digital newspaper," in *10th ACM Conference on Recommender Systems (RecSys 2016) Workshop on Profiling User Preferences for Dynamic Online and Real-Time Recommendations (RecProfile 2016)*, ACM, 2016.

[4] O. Luaces, J. Díez, T. Joachims, and A. Bahamonde, "Mapping preferences into euclidean space," *Expert Systems with Applications*, vol. 42, no. 22, pp. 8588 – 8596, 2015.

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

[6] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," *CoRR*, vol. abs/1405.4053, 2014.

[7] J. L. Herlocker, J. A. Konstan, and J. Riedl, "Explaining collaborative filtering recommendations," in *Proceedings of ACM Conference on Computer Supported Cooperative Work*, pp. 241–250, ACM, 2000.

[8] M. J. Pazzani and D. Billsus, "The adaptive web," ch. Content-based Recommendation Systems, pp. 325–341, Berlin: Springer-Verlag, 2007.

[9] J. Weston, S. Bengio, and P. Hamel, "Multi-tasking with joint semantic spaces for large-scale music annotation and retrieval," *Journal of New Music Research*, vol. 40, no. 4, pp. 337–348, 2011.

[10] S. Chen, J. L. Moore, D. Turnbull, and T. Joachims, "Playlist prediction via metric embedding," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pp. 714–722, ACM, 2012.

[11] M. Gutmann and A. Hyvärinen, "Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics," *JMLR*, vol. 13, pp. 307–361, 2012.

[12] O. Celma, *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[13] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," mar 2016.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, Nov 1997.

Una red convolucional para la clasificación de las fases de sueño

1st Isaac Fernández-Varela
CITIC
Universidade da Coruña
A Coruña, España
isaac.fvarela@udc.es

2nd Elena Hernández-Pereira
CITIC
Universidade da Coruña
A Coruña, España
elena.hernandez@udc.es

3rd Diego Alvarez-Estevez
Sleep Center & Clinical Neurophysiology
Haaglanden Medisch Centrum
The Hague, The Netherlands
diego.alvarez@udc.es

4th Vicente Moret-Bonillo
CITIC
Universidade da Coruña
A Coruña, España
vicente.moret@udc.es

***Resumen*—La clasificación de fases de sueño es una tarea crucial en el contexto de los estudios de sueño que incluye el análisis de varias señales simultáneamente y, por tanto, es tediosa y compleja. Incluso para un experto entrenado puede costar varias horas anotar las señales registradas durante el sueño de una única noche. Para resolver este problema se han desarrollado métodos automáticos, la mayor parte basándose en características inherentes al conjunto de datos utilizado. En este trabajo evitamos esa imparcialidad resolviendo el problema con un modelo de *deep learning* que puede aprender las características relevantes de las señales del paciente sin nuestra intervención. En concreto, proponemos un *ensemble* de 5 redes convolucionales que obtiene un índice kappa de $0,83$ clasificando 500 registros polisomnográficos.**

***Palabras clave*—red convolucional, fases de sueño, clasificación**

I. Introducción

Los trastornos del sueño afectan a una parte mayoritaria de la población. Como ejemplo, el 20% de los adultos españoles sufren insomnio, y entre el 12% y el 15% somnolencia diurna [1, 2]. Dormir bien es esencial para tener buena salud y las consecuencias de la falta de sueño son bien conocidas [3]. Para el diagnóstico de los trastornos del sueño es útil la identificación de las diferentes fases que atraviesa el sueño del paciente. Con este objetivo, la técnica más utilizada es el polisomnograma (PSG) que registra las señales biomédicas del paciente, entre las que se encuentran señales pneumológicas, electrofisiológicas e información contextual. Este análisis es caro, incómodo para el paciente y de difícil interpretación. Una manera habitual de simplificar esta interpretación es con el uso del hipnograma, la representación ordenada de la evolución de las fases de sueño.

El estándar de oro para la construcción del hipnograma es la guía de la *American Academy of Sleep Medicine* (AASM) [4] para la identificación de fases del sueño y sus eventos asociados, los despertares, los movimientos y los eventos cardíacos y respiratorios. Dicha guía identifica cinco fases de sueño: Despierto (*Awake*, W), movimientos oculares rápidos (*Rapid Eye Movements*, REM), y tres fases de sueño lento o no REM (N1, N2 y N3). Un hipnograma construido correctamente facilita encontrar problemas y diagnosticar trastornos del sueño, permitiendo enfocar el tiempo en la terapia. Su construcción implica analizar una gran cantidad de información y conocimiento [5]. Además, pese a las guías el acuerdo entre expertos es usualmente inferior al 90%. Por ejemplo, Stepnowsky et al. [6] estudiaron el acuerdo entre dos expertos obteniendo índices kappa entre 0,48 y 0,89. De manera similar, Wang et al. [7] obtuvieron índices entre 0,72 y 0,85. A mayores, el acuerdo también es menor para fases concretas, siendo la fase N1 la que obtiene los peores resultados.

Por todo ello, la automatización de la clasificación de fases de sueño es una tarea necesaria. La mayor parte de los métodos a día de hoy siguen una aproximación de dos pasos. Primero, extraen características específicas para este problema, muchas veces dependientes de los datos utilizados. Después, construyen un vector de características con el que se entrena algún clasificador y predicen las fases de sueño. Algunos autores utilizan un único canal de una señal y otros utilizan múltiples canales, construyendo un vector de varios elementos. Las características extraídas pueden pertenecer tanto al dominio del tiempo como al de la frecuencia. Aunque algunos trabajos utilizan una única señal, en este caso siempre el electroencefalograma (EEG), otros utilizan varias, incluyendo electrooculograma (EOG) o electromiograma (EMG), para adaptarse a las guías de la AASM.

Entre los métodos que utilizan extracción de características para su posterior clasificación encontramos los siguientes: Fraiwan et al. [8], utilizan un *random forest* para la clasificación de características del dominio del tiempo-frecuencia y las características de entropía de Reny; Liang et al. [9], obtienen la entropía en múltiples escalas y características autoregresivas analizándolas con un discriminante linear; Has-

* Esta investigación ha sido financiada en parte por la Xunta de Galicia (ED431G/01) y la Unión Europea a través del fondo ERDF.

san and Bhuiyan [10], utilizan una única señal con transformaciones *wavelet* para la extracción de características y un *random forest* para la clasificación. Sharma et al. [11], también comparan varios clasificadores, utilizando filtros iterativos para analizar un único canal de EEG; Koley and Dey [12], entrenan una *support vector machine (SVM)* con características de frecuencia, de tiempo y no lineares extraídas de un único canal de EEG; Lajnef et al. [13], utilizan varias señales y múltiples SVM para crear un árbol de decisión; Huang et al. [14], estudian la densidad espectral de potencia de 2 canales de EEG para clasificar las características de frecuencia utilizando una modificación de una SVM; Finalmente, Günes et al. [15], también analizan la densidad espectral de potencia pero clasificando con un algoritmo de *nearest neighbords*.

Solucionar el problema de clasificación de fases de sueño con extracción de características provoca sesgos por el diseño de características basadas en un único conjunto de datos. Por ello, las propuestas anteriores no generalizaban bien, concretamente dada la naturaleza de los registros de PSG, que presentan variaciones debido al paciente junto con las que provoca el hardware o el método de adquisición utilizado.

Una alternativa para resolver este problema es utilizar métodos que puedan aprender de los datos, evitando el sesgo humano. En este sentido, la apuesta natural es el *deep learning* ya que ha demostrado mejoras frente a métodos tradiciones en múltiples campos en general y en el diagnóstico médico en particular [16, 17].

Ya existen trabajos que exploran distintos modelos de *deep learning*: Längkvist et al. [18], utilizan redes *deep belief* para aprender una representación probabilística de señales preprocesadas de PSG; Tsinalis et al. [19], extraen características de una señal EEG y después utilizan redes convolucionales para la clasificación. Los mismos autores en otro trabajo [20] utilizan una pila de *sparse autoencoders*; Supratak et al. [21], utilizan una red convolucional con una red recurrente bidireccional para clasificar directamente a partir de las señales; Biswal et al. [22], comparan una red recurrente contra otros modelos, pero entrenados con características en vez de con la propia señal; Finalmente, Sors et al. [23] también utilizan una red convolucional sobre un único canal de EEG.

En este trabajo se utiliza *deep learning* para clasificar las fases de sueño a través de una red convolucional que puede aprender las características relevantes de cada fase. Siguiendo las guías de la AASM utilizamos múltiples señales; en particular, dos canales de EEG, un canal de EMG y ambos canales de EOG (derecho e izquierdo). Además, las señales se filtran previamente, para reducir el ruido y eliminar artefactos.

II. Materiales

El desarrollo y análisis del modelo que se presenta se realiza utilizando registros reales de pacientes. Dichos registros pertenecen al *Sleep Heart Health Study* (SHHS) [24], una base de datos que ofrece la *Case Western University* que proviene de un estudio entre varios centros dirigido por el *National Heart Lung and Blood Institute* para determinar las consecuencias cardiovasculares de los trastornos de sueño asociados a la respiración.

Cada registro incluye las anotaciones de distintos eventos, realizadas por expertos siguiendo las reglas de la AASM [25]. Todos los registros se anonimizaron y anotaron a ciegas. El montaje utilizado para la adquisición de las señales incluye entre otras señales dos derivaciones de EEG (C4A2 y C4A1), EOG derecho e izquierdo, un EMG submental, electrocardiograma(ECG). El EEG, EOG y EMG están muestreados a 125 Hz mientras que los EOG están a 50 Hz. Las señales se filtraron durante su adquisición con un filtro paso alto a 0,15 Hz.

Usamos tres conjuntos de datos distintos para entrenar, validar y testear nuestro modelo. El conjunto de entrenamiento incluye 400 registros, el de validación 100 y el de test 500. La duración de los registros de entrenamiento se iguala (se incluyen 7 horas aleatorias por registro) para facilitar la codificación del algoritmo de entrenamiento del modelo. De esta manera, tenemos $288,000$ ejemplos para entrenar, $119,121$ para la validación y $606,981$ para el test. Los registros se escogieron de manera aleatoria incluyendo aquellos con mucho ruido o muchos artefactos.

Las distribuciones de las diferentes clases, tanto para el conjunto de datos entero como para cada registro individual se muestran en la Tabla I. Esta tabla demuestra el desbalanceo de los conjuntos de datos, siendo la fase W predominante (aproximadamente el 38% de los casos), aunque la proporción es similar para la fase N2 (aproximadamente el 36%). Por el contrario, la clase N1 solo está representada en un 3%. También es interesante destacar que algunos registros no tienen ninguno de los casos para alguna clase y lo mucho que varían las proporciones entre ellos. Por ejemplo, en el conjunto de test, mientras un registro contiene un 7,10% para la clase N2, otro contiene un 83,43%. De hecho, estos son los dos principales problemas cuando se clasifican fases de sueño: 1) el gran desbalanceo de las clases y 2) las diferencias entre registros individuales.

III. Método

A. Filtrado de las señales

Las señales son procesadas para eliminar ruido y artefactos comunes. Ambas operaciones son pasos habituales en trabajos previos utilizados antes de la extracción de características.

El primero de los dos filtros utilizados para eliminar ruido es un filtro *Notch* centrado en 60 Hz para eliminar la interferencia que causa la línea de corriente. Este filtro se aplica a las señales con un muestreo superior a 60 Hz, EEG y EMG. El segundo elimina las frecuencias no relacionadas con movimientos musculares del EMG, utilizando un filtro paso alto a 15 Hz.

En cuanto a los artefactos, la mayor parte ocurren durante períodos muy concretos y cortos de tiempo, haciendo que sea incluso difícil reconocerlos. De cualquier manera, los artefactos ECG, causados por las interferencias del latido del corazón, son comunes y constantes a lo largo de toda la señal. Eliminamos estos artefactos usando un filtro adaptativo. Para ello, primero obtenemos la serie de latidos utilizando un

Tabla I
Distribución de las distintas clases en los conjuntos de entrenamiento, validación y test.

		W	N1	N2	N3	REM	Total
Conjunto de entrenamiento	Total	187.513	17.283	172.451	44.454	62.168	483.869
	Proporción	38,75 %	3,57 %	35,64 %	9,19 %	12,85 %	100 %
	Min en un registro	8,20 %	0,00 %	12,59 %	0,00 %	0,00 %	
	Max en un registro	71,61 %	13,75 %	68,65 %	33,43 %	26,58 %	
Conjunto de validación	Total	43.742	3.963	43.510	12.900	15.006	119.121
	Proporción	36,72 %	3,33 %	36,53 %	10,83 %	12,60 %	100 %
	Min en un registro	11,21 %	0,29 %	12,38 %	0,00 %	0,00 %	
	Max en un registro	76,79 %	17,08 %	60,09 %	30,16 %	23,68 %	
Conjunto de test	Total	231.707	19.769	217.246	61.281	76.978	606.981
	Proporción	37,77 %	3,26 %	35,96 %	10,25 %	12,75 %	100 %
	Min en un registro	7,75 %	0,00 %	7,10 %	0,00 %	0,00 %	
	Max en un registro	76,53 %	16,93 %	83,43 %	43,82 %	31,11 %	

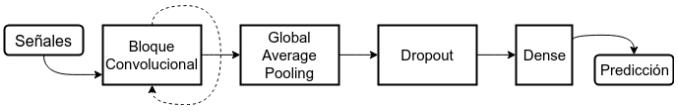


Figura 1. Red convolucional propuesta.

algoritmo estándar de detección de ondas QRS [26]. Después estudiamos la calidad de la señal de ECG para saber que intervalos podemos incluir en la construcción del filtro adaptativo. Por último, durante los intervalos con suficiente calidad, aplicamos y actualizamos el filtro adaptativo para eliminar los artefactos. Este proceso se describe en profundidad en Fernández-Varela et al. [27].

B. Red convolucional

La clasificación de las fases de sueño se realiza habitualmente con ventanas de 30 s, llamadas epochs. Analizando múltiples características de cada epoch, los expertos clínicos pueden decidir cual es la fase de sueño correspondiente a dicho epoch.

Una red convolucional [28] es una red *deep-forward* que soluciona las limitaciones del perceptrón multicapa con una arquitectura que comparte pesos. Básicamente, aplica una operación de convolución sobre la entrada, reduciendo el número de parámetros. Por eso, permite construir redes más profundas que facilitan reconocer características más complejas. La red propuesta se representa en la Figura 1.

La red convolucional recibe como entrada el conjunto de señales (2 canales de EEG, EMG y ambos EOG). Cada entrada es un epoch: ventana de 30 segundos. Debido a la diferencia de muestreo entre las señales, tal y como se comentó en la Sección II, se realiza una operación de *upsampling* a 125 Hz. Se descarta un *downsample* a 50 Hz porque perderíamos las frecuencias altas del EEG que clínicamente contienen información interesante para clasificar las fases de sueño. De la misma forma, se descarta una operación de *padding* porque no sería fácilmente extensible a otros conjuntos de datos con señales adquiridas con otras frecuencias de muestreo. De esta manera, cada entrada de la red convolucional es una matriz de 3750×5. Cada señal se normaliza con media 0 y desviación 1, obteniendo la media y desviación a partir de todas las señales correspondientes del conjunto de datos de entrenamiento. Usando otras normalizaciones de menor granularidad las redes probadas inicialmente no convergían. El bloque convolucional que se muestra en la Figura 1 es una sucesión de cuatro capas incluyendo una convolución 1D que preserva el tamaño de la entrada (con *padding*), una capa de *batch normalization* [29] para la regularización, una activación ReLu [30] y un *average pool* que reduce el tamaño de la entrada por un factor de 2. Usando una convolución 1D evitamos imponer una estructura espacial que desconocemos entre las distintas señales. Este bloque se repitió n veces, siendo n un hiperparámetro cuyo valor fue seleccionado durante la experimentación. Todas las capas se configuraron con el mismo tamaño de kernel pero el número de filtros para la capa i es dos veces el número de filtros de la capa $i-1$. La selección de n, el tamaño de kernel y el número inicial de filtros se explica en la siguiente sección, junto a otros hiperparámetros.

La salida del último bloque convolucional, tras ajustar la dimensión con un *global pooling* y aplicarle *dropout* para mejorar la regularización, se utiliza como entrada en una capa densa con activación *softmax*. Esta capa devuelve la probabilidad de cada clase para la entrada. Como es habitual, se selecciona la clase con mayor probabilidad como decisión de clasificación.

Para entrenar la red se utiliza el optimizador Adam [31] con 64 elementos por *batch*. Este tamaño de *batch* está condicionado por el hardware disponible para la ejecución. Del optimizador se configura el hiperparámetro ratio de aprendizaje, manteniendo ambas betas con los valores por defecto. El entrenamiento se termina utilizando *early stopping* monitorizando la pérdida en el conjunto de validación con una paciencia de 10 epochs. Debido al desbalanceo de las clases se utiliza *cross entropy* ponderada, obteniendo los pesos del conjunto de entrenamiento.

C. Optimización de hiperparámetros

Una correcta elección de los hiperparámetros puede significar el éxito de un modelo de*deep learning*. La dificultad a

la hora de seleccionar los mejores hiperparámetros no es sólo obtener el mejor rendimiento sino en conseguirlo minimizando el coste al hacerlo, pudiendo ser este coste económico o computacional.

En este trabajo se utiliza un *Tree-structured Parzen Estimator* (TPE) que se ha mostrado superior frente a otros métodos [32, 33]. El TPE es un aproximación secuencial de optimización basada en modelos (SMBO). Los métodos SMBO construyen secuencialmente modelos para aproximar el rendimiento de una selección de hiperparámetros basándose en resultados históricos y así escoger nuevos hiperparámetros que se comprueban con el modelo. Particularmente, TPE modela dos distribuciones $P(x|y)$ y $P(y)$ donde x representa los hiperparámetros e y el rendimiento asociado, y optimiza la mejora esperada (*expected improvement*, EI) siguiendo la ecuación:

$$EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)\frac{P(x|y)P(y)}{P(x)}$$

donde y^* es algún cuantil γ de los valores observados y tal que $p(y < y^*) = \gamma$.

Utilizamos TPE para optimizar los siguientes hiperparámetros relacionados con la red convolucional: el número de bloques convolucionales, el tamaño del kernel de cada convolución 1D y el número de filtros del primer bloque. A mayores, existe una relación entre el número inicial de filtros y el número de bloques convolucionales. Dadas nuestras restricciones computacionales no añadimos bloques que tuviesen más de 1024 filtros.También se utiliza TPE para el ratio de aprendizaje del optimizador. Las distribuciones usadas para cada uno de estos hiperparámetros se resumen en la Tabla II.

Tabla II
Distribuciones para los distintos hiperparámetros

Hiperparámetro	Distribución
Bloques convolucionales	Uniforme entre 1 y 10
Tamaño del kernel	Uniforme entre 3 y 50
Filtros iniciales	Elección entre 8, 16, 32 o 64
Ratio de aprendizaje	Log-uniforme entre -10 y -1

Para reducir el tiempo computacional de selección de los hiperparámetros utilizamos un subconjunto del conjunto de entrenamiento para entrenar, validar y hacer el test de los distintos modelos. Este subconjunto contiene 250 registros de los que 20 se utilizan para validación durante el entrenamiento y 50 para el test de cada modelo. En total probamos 50 modelos utilizando el índice kappa para seleccionar el mejor.

D. Medidas de rendimiento

El rendimiento de los modelos se evalúa con las siguientes medidas:

- **Precisión,** la fracción entre el número de verdaderos positivos y el número de predicciones positivas.
- **Sensitividad**, la fracción entre el número de verdaderos positivos y el número elementos de la clase.

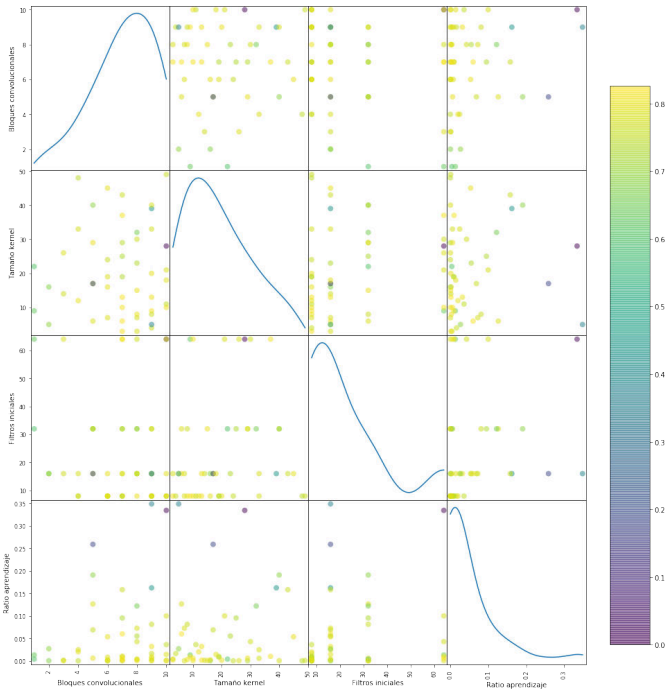


Figura 2. Gráfico de dispersión para los diferentes conjuntos de hiperparámetros probados. El color del punto representa el índice kappa para el modelo con dichos hiperparámetros. La diagonal representa la distribución de los valores asignados para ese hiperparámetro concreto.

- ***F1 score***, la media armónica de precisión y sensitividad.
- **Kappa**, la medida del acuerdo entre dos clasificadores que tiene en cuenta la posibilidad de acuerdo casual. El acuerdo perfecto obtiene un valor de 1 y solo por casualidad un valor 0.

IV. Resultados

Antes de ver los resultados conseguidos podemos visualizar el rendimiento que obtuvieron los distintos modelos probados para la búsqueda de hiperparámetros en la Figura 2. Destaca la necesidad de un ratio de aprendizaje bajo para que el entrenamiento converja adecuadamente.

Para mejorar los resultados obtenidos por nuestro modelo utilizamos un *ensemble*. Varios modelos clasifican las fases de sueño de manera individual y el resultado final es la fase más repetida. En este caso, hemos escogido los 5 mejores modelos obtenidos durante la selección de hiperparámetros. Los valores para los hiperparámetros de cada uno de ellos se presentan en la Tabla II.

Los resultados obtenidos por el *ensemble* utilizando el conjunto de test se muestran en la Tabla IV. La mejor clasificación se obtiene para la clase W, con valores cercanos a $0,95$ para la precisión, sensitividad y *F1 score*; después las clases N2, N3 y REM presentan resultados similares, especialmente si nos fijamos en la *F1 score*, aunque con menor sensitividad para la clase N3 y, por tanto, también mayor precisión. Por último, los resultados no son los esperados para la clasificación de la fase N1, no llegándose a un *F1 score* de $0,3$. Típicamente, N1 es la clase más difícil de clasificar incluso para los expertos.

Tabla III
Hiperparámetros para los 5 mejores modelos

Parámetro	Modelo 1	Modelo 2	Modelo 3	Modelo 4	Modelo 5
Bloque convolucionales	7	9	7	7	7
Tamaño del kernel	6	9	13	3	10
Filtros iniciales	16	8	8	8	64
Ratio de aprendizaje	$5,99 \times 10^{-2}$	$9,00 \times 10^{-3}$	$1,45 \times 10^{-3}$	$1,91 \times 10^{-3}$	$5,49 \times 10^{-3}$

Tabla IV
Medidas de rendimiento para la clasificación del conjunto de test utilizando el ensemble de los 5 mejores modelos.

Fase	Precisión	Sensitividad	*F1 score*
W	0,94	0,96	0,95
N1	0,39	0,21	0,27
N2	0,87	0,89	0,88
N3	0,92	0,77	0,84
REM	0,82	0,90	0,86
Average	0,78	0,75	0,76

La matriz de confusión obtenida con el *ensemble* se muestra en la Figura 3, en la que se puede comprobar que la mayor parte de las fases N1 son clasificadas como otras fases, especialmente N2. Además, aunque en una proporción mucho menor, cuando se comete un error clasificando una clase distinta a N2, también se tiende a clasificar como N2.

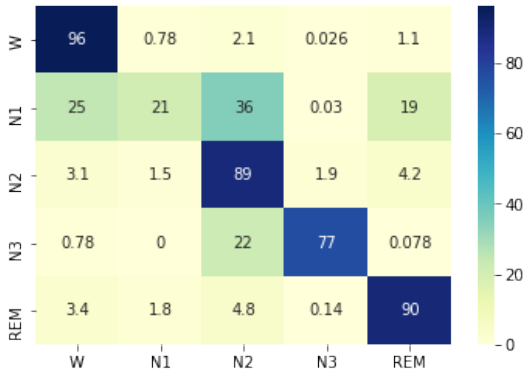


Figura 3. Matriz de confusión para la clasificación del conjunto de test utilizando el *ensemble* de los 5 mejores modelos.

V. Discusión y Conclusiones

En este trabajo presentamos un *ensemble* de redes convolucionales para la clasificación de fases de sueño. Es una tarea que consume mucho tiempo y crítica a la hora de diagnosticar trastornos del sueño. La mayor parte de los métodos automáticos de clasificación de fases de sueño se basan en características diseñadas para un conjunto de datos concreto y no suelen, por tanto, adaptarse bien a otros conjuntos de datos. Para solucionar este problema usamos una red convolucional que puede aprender las características relevantes de las señales involucradas en la tarea por su cuenta.

Un aspecto importante para el éxito o el fracaso de los modelos convolucionales es la elección correcta de los hiperparámetros. En nuestro caso concreto trabajamos con 4 hiperparámetros, optimizando su selección con un *tree-structured parzen estimator* y evaluando 50 modelos distintos.

El *ensemble* propuesto, a partir de las 5 mejores configuraciones de hiperparámetros, obtiene una precisión, sensitividad y un *F1 score* medias de $0,78$, $0,75$ y $0,76$ respectivamente, con un índice kappa de $0,83$. Aunque globalmente los resultados son aceptables, nuestra solución ha mostrado problemas para clasificar la fase N1. Además, cuando la clasificación es incorrecta suele ser hacia la clase N2.

La comparación frente a trabajos similares es difícil por la falta de estándares, tanto en los conjuntos de datos como en el proceso de evaluación. En la Tabla V mostramos los resultados de trabajos anteriores, limitándonos a los que ofrecen valores para cada clase. Como se puede ver, obtenemos el índice kappa más alto, aunque no los mejores *F1 score*. Salvo para la clase W, algún trabajo presenta mejor *F1 score*. De cualquier manera, los valores que obtenemos son competitivos, con la excepción de la clase N1, aunque queda claro en la comparación que dicha clasificación es la más difícil. Frente al único trabajo que presenta resultados con un conjunto de datos similar [23], obtenemos mejor índice Kappa y *F1 score* para la clase W, con valores casi idénticos para N2, N3 y REM aunque bastante más bajos para N1.

Los resultados son prometedores y el método escogido debería ser adaptable a otros conjuntos de datos, especialmente si además se puede adaptar el entrenamiento. Además, entrenando simultánemante con varios conjuntos de datos la red debería generalizar mejor, evitando especializarse en registros concretos.

Para mejorar estos resultados es necesario explicar como se hace la clasificación. Además, sería interesante introducir memoria en el modelo, posiblemente en forma de red recurrente, porque en ciertas condiciones, la clasificación de un epoch depende de los epochs anteriores.

Bibliografía

[1] M. M. Ohayon and T. Sagales, "Prevalence of insomnia and sleep characteristics in the general population of spain." *Sleep medicine*, vol. 11, no. 10, pp. 1010–8, dec 2010.

[2] J. Marin *et al.*, "Prevalence of sleep apnoea syndrome in the spanish adult population," *International Journal of Epidemiology*, vol. 26, no. 2, pp. 381–386, apr 1997.

[3] H. R. Colten and B. M. Altevogt, *Sleep Disorders and Sleep Deprivation*. Washington, D.C.: National Academies Press, sep 2006, vol. 6, no. 9.

Tabla V
Comparación de resultados de trabajos previos para la clasificación de fases del sueño.

Trabajo	Conjunto de datos	Kappa	F1 score				
			W	N1	N2	N3	REM
Biswal et al. [22]	Massachusetts General Hospital, 1000 registros	0,77	0,81	**0,70**	0,77	0,83	**0,92**
Längkvist et al. [18]	St Vicent's University Hospital, 25 registros	0,63	0,73	0,44	0,65	**0,86**	0,80
Sors et al. [23]	SHHS, 1730 registros	0,81	0,91	0,43	0,88	0,85	0,85
Supratak et al. [21]	MASS dataset, 62 registros	0,80	0,87	0,60	**0,90**	0,82	0,89
Supratak et al. [21]	SleepEDF, 20 registros	0,76	0,85	0,47	0,86	0,85	0,82
Tsinalis et al. [19]	SleepEDF, 39 registros	0,71	0,72	0,47	0,85	0,84	0,81
Tsinalis et al. [20]	SleepEDF, 39 registros	0,66	0,67	0,44	0,81	0,85	0,76
This work	SHHS, 500 registros	**0,83**	**0,95**	0,27	0,88	0,84	0,86

[4] R. B. Berry *et al.*, "AASM Scoring Manual Updates for 2017 (Version 2.4)." *Journal of clinical sleep medicine : JCSM : official publication of the American Academy of Sleep Medicine*, vol. 13, no. 5, pp. 665–666, may 2017.

[5] Á. Fernández-Leal *et al.*, "A knowledge model for the development of a framework for hypnogram construction," *Knowledge-Based Systems*, vol. 118, pp. 140–151, 2017.

[6] C. Stepnowsky *et al.*, "Scoring accuracy of automated sleep staging from a bipolar electroocular recording compared to manual scoring by multiple raters." *Sleep medicine*, vol. 14, no. 11, pp. 1199–207, nov 2013.

[7] Y. Wang *et al.*, "Evaluation of an automated single-channel sleep staging algorithm." *Nature and science of sleep*, vol. 7, pp. 101–11, 2015.

[8] L. Fraiwan *et al.*, "Automated sleep stage identification system based on time–frequency analysis of a single EEG channel and random forest classifier," *Computer Methods and Programs in Biomedicine*, vol. 108, no. 1, pp. 10–19, oct 2012.

[9] J. Liang *et al.*, "Predicting seizures from electroencephalography recordings: A knowledge transfer strategy," in *Proceedings - 2016 IEEE International Conference on Healthcare Informatics, ICHI 2016*. IEEE, oct 2016, pp. 184–191.

[10] A. R. Hassan and M. I. H. Bhuiyan, "A decision support system for automatic sleep staging from EEG signals using tunable Q-factor wavelet transform and spectral features," *Journal of Neuroscience Methods*, vol. 271, pp. 107–118, sep 2016.

[11] R. Sharma, R. B. Pachori, and A. Upadhyay, "Automatic sleep stages classification based on iterative filtering of electroencephalogram signals," *Neural Computing and Applications*, vol. 28, no. 10, pp. 2959–2978, oct 2017.

[12] B. Koley and D. Dey, "An ensemble system for automatic sleep stage classification using single channel EEG signal," *Computers in Biology and Medicine*, vol. 42, no. 12, pp. 1186–1195, 2012.

[13] T. Lajnef *et al.*, "Learning machines and sleeping brains: Automatic sleep stage classification using decision-tree multi-class support vector machines," *Journal of Neuroscience Methods*, vol. 250, pp. 94–105, jul 2015.

[14] C.-S. Huang *et al.*, "Knowledge-based identification of sleep stages based on two forehead electroencephalogram channels," *Frontiers in Neuroscience*, vol. 8, p. 263, sep 2014.

[15] S. Günes, K. Polat, and S. Yosunkaya, "Efficient sleep stage recognition system based on EEG signal using k-means clustering based feature weighting," *Expert Systems with Applications*, vol. 37, no. 12, pp. 7922–7928, dec 2010.

[16] A. Esteva *et al.*, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, feb 2017.

[17] V. Gulshan *et al.*, "Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs," *JAMA*, vol. 316, no. 22, p. 2402, dec 2016.

[18] M. Längkvist *et al.*, "Sleep Stage Classification Using Unsupervised Feature Learning," *Advances in Artificial Neural Systems*, vol. 2012, pp. 1–9, 2012.

[19] O. Tsinalis *et al.*, "Automatic sleep stage scoring with single-channel eeg using convolutional neural networks," oct 2016.

[20] O. Tsinalis, P. M. Matthews, and Y. Guo, "Automatic sleep stage scoring using time-frequency analysis and stacked sparse autoencoders," *Annals of Biomedical Engineering*, vol. 44, no. 5, pp. 1587–1597, may 2016.

[21] A. Supratak *et al.*, "Deepsleepnet: A model for automatic sleep stage scoring based on raw single-channel eeg," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 25, no. 11, pp. 1998–2008, nov 2017.

[22] S. Biswal *et al.*, "Sleepnet: Automated sleep staging system via deep learning," jul 2017.

[23] A. Sors *et al.*, "A convolutional neural network for sleep stage scoring from raw single-channel EEG," *Biomedical Signal Processing and Control*, vol. 42, pp. 107–114, apr 2018.

[24] S. F. Quan *et al.*, "The sleep heart health study: Design, rationale, and methods," *Sleep*, vol. 20, no. 12, pp. 1077–1085, dec 1997.

[25] M. H. Bonnet *et al.*, "EEG arousals: scoring rules and examples: a preliminary report from the Sleep Disorders Atlas Task Force of the American Sleep Disorders Association." *Sleep*, vol. 15, no. 2, pp. 173–184, apr 1992.

[26] V. Afonso *et al.*, "Ecg beat detection using filter banks," *IEEE Transactions on Biomedical Engineering*, vol. 46, no. 2, pp. 192–202, 1999.

[27] I. Fernández-Varela *et al.*, "A simple and robust method for the automatic scoring of EEG arousals in polysomnographic recordings," *Computers in Biology and Medicine*, vol. 87, pp. 77–86, aug 2017.

[28] Y. Le Cun *et al.*, "Handwritten digit recognition: applications of neural network chips and automatic learning," *IEEE Communications Magazine*, vol. 27, no. 11, pp. 41–46, nov 1989.

[29] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[30] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," *Proceedings of the 27th International Conference on Machine Learning*, no. 3, pp. 807–814, 2010.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," dec 2014.

[32] J. Bergstra *et al.*, "Algorithms for hyper-parameter optimization," in *NIPS*, 2011.

[33] J. Bergstra, D. Yamins, and D. D. Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," in *Proc. of the 12th python in science conf*, 2013.

Evaluación de estrategias de binarización en la clasificación de imágenes usando deep learning

Francisco Pérez, Siham Tabik, Alberto Castillo, Hamido Fujita* y Francisco Herrera
Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada, Granada, España
{fperezhernandez,siham}@ugr.es,{albertocl,herrera}@decsai.ugr.es
*Iwate Prefectural University, Takizawa, Iwate,Japan

***Resumen*—El reconocimiento de objetos pequeños en imágenes con redes neuronales convolucionales (CNNs) sigue siendo un reto, especialmente cuando estos objetos se manipulan con la mano de forma muy similar. En este trabajo proponemos dividir un problema multiclase del ámbito de la seguridad en un problema de binarización de clases para obtener un mejor resultado. Nuestro objetivo ha sido obtener el mejor modelo de aprendizaje que distinga entre 6 clases, pistolas, smartphone, billete, monedero, tarjeta y background. Concretamente, evaluamos las técnicas *One-Versus-All* (OVA), *One-Versus-One* (OVO) y *Distance-based Relative Competence Weighting combination* para OVO (DRCW-OVO) basadas en CNNs. El mejor rendimiento se obtiene usando DRCW-OVO con una precisión de 90,47 %, un recall del 90,93 % y un F1 del 90,59 %. Esto significó una mejora del 2,58 % en precisión, 1,48 % en recall y 2,13 % en F1 frente a un multiclasificador normal.**

***Index Terms*—Clasificación, *Convolutional Neuronal Networks* (CNNs), Multiclasificación, *Deep Learning*, *Machine Learning*, *One-Versus-All* (OVA), *One-Versus-One* (OVO), DRCW-OVO, ResNet-101**

I. Introducción

La tarea de clasificar imágenes es un reto en nuestros días y se puede ver en la rama de visión por computador. En competiciones como ImageNet [1], cada año participan muchos equipos de diferentes grandes empresas como Google, y la diferencia para obtener los mejores resultados es crucial.

En el paradigma de *machine learning* clásico, la clasificación es un problema bien conocido donde implementar nuevas técnicas para mejorar los resultados. Es el caso de la descomposición de un problema multiclase en problemas biclase. La clásica técnica *One-Versus-All* (OVA) y *One-Versus-One* (OVO) se usa en muchos trabajos como un buen instrumento para aumentar el rendimiento de los modelos.

ImageNet y COCO (*Common Objects in Context*) [2], abordan un problema de clasificación de imágenes. Concretamente, la tarea es diferenciar objetos cotidianos que pueden provocar una confusión entre ellos. Un problema bien conocido por nuestro grupo de investigación es la detección de armas de fuego. Este tipo de imágenes suponen un reto ya que hay muchos objetos que se pueden manejar de la misma forma.

La mayoría de trabajos anteriores en este ámbito abordaban la detección de armas en imágenes de rayos X, milimétricas o RGB utilizando métodos clásicos de *machine learning* que requieren una alta intervención humana [3], [4], [5], [6], [7].

Actualmente, los modelos más precisos en la clasificación de imágenes y detección de objetos se basan en redes neuronales convolucionales profundas (CNNs) [8]. Estos modelos aprenden automáticamente las características distintivas de los objetos a partir de un gran conjunto de datos etiquetados.

Por lo que sabemos, el primer modelo de detección automática de pistolas en vídeo basado en CNNs fue desarrollado por Olmos et al en [9]. Sin embargo, cuando en los vídeos, una persona manipula objetos como smartphone, billete, monedero o tarjeta, el modelo produce falsos positivos. La Figura 1 muestra ejemplos de este tipo de falsos positivos cometidos por el modelo de detección. Esto puede explicarse por el hecho de que el modelo aprendió la forma en que se manejan las pistolas, siendo también una característica clave, lo cual es intolerable en el campo de la videovigilancia debido a todas las posibles falsas alarmas que se puedan producir.

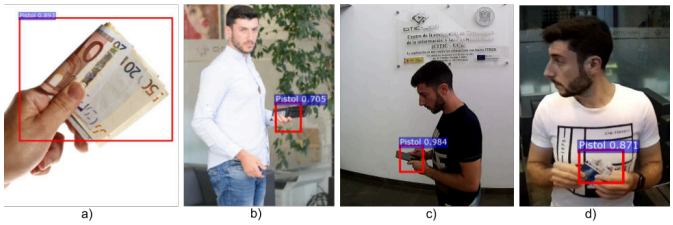


Figura 1. Falsos positivos cometidos por el modelo de detección, donde se confunde con a) billete, b) monedero, c) smartphone y d) tarjeta.

Para mejorar la precisión y robustez del modelo, (1) construimos un conjunto de datos de entrenamiento de alta calidad que incluye todas las clases de objetos posibles que se manejan comúnmente de manera similar, (2) desarrollamos un modelo de clasificación robusto y (3) mejoramos la robustez usando técnicas de *machine learning* como OVA y OVO [10]. Además, el método *Distance-based Relative Competence Weighting combination* para OVO (DRCW-OVO) [11], para problemas multiclase, es utilizado como una extensión de los métodos clásicos de agregación de OVO. Por lo tanto, nos centramos en la tarea de clasificación, ya que es la base de la detección. De hecho, los modelos de detección más influyentes combinan un modelo de clasificación con un método de búsqueda de regiones [12], [13], [14].

Este trabajo se organiza analizando, en la Sección II, los preliminares, donde se analizan trabajos relacionados y las

estrategias de descomposición en problemas multiclase. En la sección III la construcción de la base de datos. La evaluación de las estrategias usadas se encuentra en la sección IV y finalmente las conclusiones aparecen en la sección V.

II. Preliminares

La mayoría de los trabajos analizados hacen uso de OVA y OVO en tareas visuales, reconocimiento de objetos, clasificación de imágenes y segmentación de imágenes, utilizando sólo modelos clásicos como *Support Vector Machine* (SVM), *Linear Discriminant Analysis* (LDA) y *k-Nearest Neighbors* (k-NN). Por ejemplo, en clasificación de imágenes, los autores en [15] analizaron el enfoque OVA y OVO para reducir el espacio de las características en tres dataset bien conocidos, MNIST, *Amsterdam Library of Object Images* (ALOI) y *Australian Sign Language* (Auslan). Para la estimación de la pose en la segmentación de imágenes, los autores en [16] compararon un clasificador individual basado en CNN con OVA y OVO basado en SVM y mostraron que las CNN logran un rendimiento ligeramente mejor que OVA y OVO basados en SVM. De manera similar, en la tarea de clasificación de imágenes *remote sensing*, los autores en [17] también compararon OVA y OVO basados en SVM y 1-NN y concluyeron que OVA proporcionó peores resultados debido al desequilibrio entre clases. Los mejores resultados fueron obtenidos por OVO con SVM. En el reconocimiento facial, los autores de [18] utilizaron un modelo basado en CNN para la extracción de características y un SVM, OVA y OVO para la clasificación. Los mejores resultados fueron obtenidos por CNN en combinación con SVM. Los autores en [19] compararon la técnica *Half-Against-Half* (HAH) con OVA y OVO en la clasificación de imágenes y encontraron que HAH proporciona resultados similares o peores en los puntos de referencia evaluados. Nuestro trabajo se diferencia de todos los anteriores en que mejora la robustez del reconocimiento de objetos que se manejan similarmente utilizando OVA, OVO y DRCW-OVO basados en CNNs. Hasta donde sabemos, ningún trabajo previo aplicó estas técnicas en modelos de Deep Learning para la clasificación de imágenes.

Los problemas de clasificación que involucran múltiples clases son más difíciles de resolver. El enfoque común para abordar este tipo de problemas es reformular el problema original multiclase en un conjunto de problemas binarios de dos clases. Las técnicas más comunes en este contexto son OVA [20] y OVO [21]. En [22] se ofrece una explicación ampliada y completa de todos los posibles métodos de agregación para OVA y OVO. Además, el método DRCW-OVO [11] amplía el clásico OVO utilizando la distancia entre clases.

II-A. OVA

La estrategia *One-Versus-All* (OVA) reformula el problema de la clasificación multiclase en un conjunto de clasificadores binarios donde cada clasificador aprende cómo distinguir cada clase individual contra el resto de clases juntas. Este enfoque produce tantos clasificadores como el número de clases en el problema original. La predicción final se calcula combinando las predicciones de los clasificadores individuales mediante un método de agregación denominado *Maximum Confidence Strategy* (MAX). La clase con el mayor número de votos se considera como la clase pronosticada. Formalmente,

$$PredictedClass = arg \max_{i=1,\ldots,m} r_i$$

, donde $r_i \in [0,1]$ es la confianza para la clase i y m es el número de clases. En la Figura 2 se ilustra OVA aplicado al problema multiclase considerado en este trabajo.

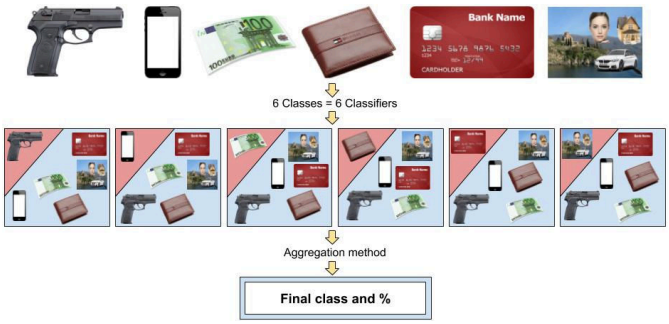


Figura 2. Proceso OVA.

II-B. OVO

La estrategia *One-Versus-One* (OVO) reformula el problema multiclase original en tantos problemas binarios como combinaciones posibles entre pares de clases para que cada clasificador aprenda a discriminar entre cada par. Es decir, un problema $m-$clases se convertirá en $m(m-1)/2$ clasificadores y en el caso considerado, con m = 6 clases, se reformulará en 15 clasificadores, como se aprecia en la Figura 3.

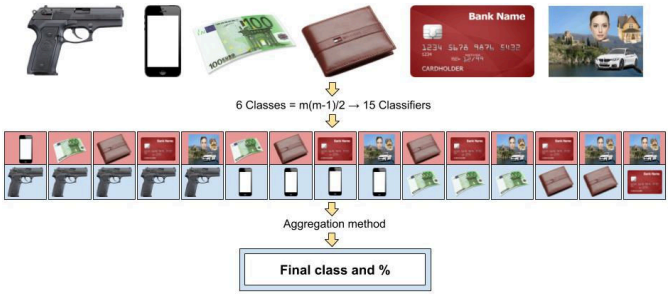


Figura 3. Proceso OVO.

El sistema OVO puede utilizar diversas estrategias de agregación como pueden ser *Max-Wins* (VOTE aleatorio o por peso), *Weighted Voting Strategy* (WV), *Learning Valued Preference for Classification* (LVPC), *Preference Relations Solved by Non-Dominance Criterion* (ND), *Classification by Pairwise Coupling* (PC) y *Wu, Lin and Weng Probability Estimates by Pairwise Coupling approach* (PE).

VOTE random en OVO: La regla VOTE, también llamada regla *Max-Wins* [23], se considera como la regla básica de decisión en OVO. En este método se repasa la matriz y en el elemento r_{ij} si la posibilidad de pertenecer a la clase i es superior a 0,5, el resultado es la clase i. Finalmente, se repasa

toda la matriz, se suma el resultado y se selecciona la clase con más votos. Si tenemos dos o más clases con el mismo número de votos, seleccionamos una al azar. Formalmente,

$$PredictedClass = arg \max_{i=1,...,m} \sum_{i \le j \ne i \le m} s_{ij}$$

donde s_{ij} es 1 si $r_{ij} > r_{ji}$ y 0 en caso contrario.

VOTE weight en OVO: Este es un nuevo enfoque propuesto. En el método VOTE, podemos tener 2 o más clases con los mismos votos. Para las clases con mayor número de votos, proponemos sumar las predicciones y seleccionar la clase con el valor máximo como la clase final.

WV en OVO: La técnica *Weighted Voting strategy* pretende obtener la clase con la mayor probabilidad. Por esta razón, cada clase suma sus predicciones y la clase con el valor máximo es el resultado final. La regla de decisión es:

$$PredictedClass = arg \max_{i=1,...,m} \sum_{i \le j \ne i \le m} r_{ij}$$

LVPC en OVO: Learning Valued Preference for Classification (LVPC) [24], [25] es la técnica que utiliza el peso de las clases y que penaliza a las clases que no tienen suficiente certeza. La regla de decisión es:

$$P_{ij} = r_{ij} - min\{r_{ij}, r_{ji}\}; P_{ji} = r_{ji} - min\{r_{ij}, r_{ji}\}$$

$$C_{ij} = min\{r_{ij}, r_{ji}\}; I_{ij} = 1 - max\{r_{ij}, r_{ji}\}$$

$$Class = arg \max_{i=1,...,m} \sum_{i \le j \ne i \le m} P_{ij} + \frac{1}{2} C_{ij} + \frac{N_i}{N_i + N_j} Iij$$

donde N_i es el número de ejemplos de la clase i en train.

ND en OVO: La técnica *Preference Relations Solved by Non-Dominance Criterion* (ND) se definió originalmente para la toma de decisiones con relaciones de preferencia difusas [26]. En [27] se aplica el mismo criterio en un sistema de clasificación OVO. Primero normalizando, seguido de calcular la preferencia difusa y calcular el grado para cada clase, para obtener la clase final:

$$\overline{r}_{ij} = \frac{r_{ij}}{r_{ij} + r_{ji}}$$

$$r'_{ij} = \begin{cases} \overline{r}_{ij} - \overline{r}_{ji}, & cuando\ \overline{r}_{ij} > \overline{r}_{ji} \\ 0, & de\ otra\ manera \end{cases}$$

$$ND_i = 1 - \sup_{j \epsilon C}[r'_{ji}]$$

$$Class = arg \max_{i=1,...,m} ND_i$$

PC en OVO: La técnica *Classification by Pairwise Coupling* (PC) [28] intenta mejorar la estrategia de votación cuando los resultados de los clasificadores son probabilidades estimadas de clase. Este método estima la probabilidad conjunta para todas las clases a partir de las probabilidades de clase por pares de los clasificadores binarios. El algoritmo es:

1. Inicialización:

$$\widehat{p}_i = \frac{2}{m} \frac{\sum_{1 \le j \ne i \le m} r_{ij}}{(m-1)}\ para\ todo\ i = 1, ..., m$$

$$\widehat{\mu}_{ij} = \frac{\widehat{p}_i}{\widehat{p}_i + \widehat{p}_j}\ para\ todo\ i, j = 1, ..., m$$

2. Repetir hasta converger:

$$\widehat{p}_i = \widehat{p}_i \frac{\sum_{1 \le j \ne i \le m} n_{ij} r_{ij}}{\sum_{1 \le j \ne i \le m} n_{ij} \widehat{\mu}_{ij}}\ para\ todo\ i = 1, ..., m$$

donde n_{ij} es el número de elementos en train en las clases ith y jth.

$$\widehat{p}_i = \frac{\widehat{p}_i}{\sum_{i=1} \widehat{p}_i}\ para\ todo\ i = 1, ..., m$$

$$\widehat{\mu}_{ij} = \frac{\widehat{p}_i}{\widehat{p}_i + \widehat{p}_j}\ para\ todo\ i, j = 1, ..., m$$

Finalmente, la salida de la clase será:

$$Class = arg \max_{i=1,...,m} \widehat{p}_i$$

PE en OVO: La técnica *Wu, Lin y Weng Probability Estimates by Pairwise Coupling Approach* (PE) [29] es similar a PC, estima las probabilidades (p) de cada clase a partir de las probabilidades por pares. PE optimiza el siguiente problema:

$$\min_p \sum_{i=1}^{m} \sum_{1 \le j \ne i \le m} (r_{ji} p_i - r_{ij} p_j)^2\ sujeto\ a$$

$$\sum_{i=1}^{k} p_i = 1, p_i \ge 0, \forall i$$

II-C. DRCW-OVO

Distance-based Relative Competence Weighting combination para *One-Versus-One* (DRCW-OVO) en problemas multiclase [11] es una extensión de la técnica OVO que pretende mejorar el problema del imbalanceo de las clases usando la distancia con los k elementos vecinos a la nueva instancia.

DRCW-OVO, una vez que se ha obtenido la matriz de pesos:

1. Calcular la distancia media de los k vecinos cercanos a cada clase en el vector **d**.
2. Calcular la nueva matriz de pesos R^w de la forma:

$$r^w_{ij} = r_{ij} \cdot w_{ij} \text{ donde } w_{ij} = \frac{d_j^2}{d_i^2 + d_j^2}$$

$$r'_{ij} = \begin{cases} \overline{r}_{ij} - \overline{r}_{ji}, & cuando\ \overline{r}_{ij} > \overline{r}_{ji} \\ 0, & de\ otra\ manera \end{cases}$$

siendo d_i la distancia de la instancia a los vecinos cercanos de la clase i.

3. Usar la estrategia *Weighted Voting* (WV) en la nueva matriz de pesos R^w para obtener la clase final.

Se ha calculado, para la distancia entre imágenes, la distancia *Quadratic-Chi* [30] con el histograma de las imágenes:

$$X^2(P, Q) = \frac{1}{2} \sum_i \frac{(P_i - Q_i)^2}{(P_i + Q_i)}$$

donde P_i es el histograma de la nueva instancia y Q_i es la media del histograma de los k vecinos cercanos.

III. Construcción de la base de datos guiada por el rendimiento en clasificación

El objetivo de esta sección es construir un dataset que permita al modelo de clasificación distinguir entre objetos que se manejan de forma similar. El proceso de construcción se ha guiado por el rendimiento del modelo de clasificación.

Se ha usado el modelo de clasificación ResNet-101 [31] inicializándolo con los pesos preentrenados en ImageNet [1]. Como software de Deep Learning se ha usado Tensorflow [32] y los experimentos se han realizado en una NVIDIA Titan Xp, durando dos horas cada entrenamiento.

Para mejorar el aprendizaje del modelo, se ha desarrollado el conjunto de datos en cinco pasos:

1. Como punto de partida se ha usado el dataset [1] construido en [9] de imágenes de pistolas, junto a la clase background, donde se han incluido caras, coches, escenas, etc. La mayoría de las imágenes fueron descargadas de Internet.
2. A las imágenes del punto 1, se ha sumado la primera clase competitiva, el smartphone.
3. Se han añadido el resto de objetos que pueden causar confusión como billete, monedero y tarjeta.
4. Enriquecimos todas las clases del conjunto de datos con imágenes tomadas por diferentes cámaras con diversas calidades y resoluciones, una cámara réflex, Nikon D5200, y dos cámaras de videovigilancia, Hikvision DS-2CD2420F-IW y Samsung SNH-V6410PN.
5. Eliminamos las imágenes borrosas junto con las imágenes en las que el ojo humano podía confundir los distintos objetos.

Para evaluar la calidad de cada conjuntos de datos se utilizó la base de datos Dataset-Test. Las características de los conjuntos de datos construidos se proporcionan en la Tabla I.

Tabla I
Elementos que componen cada uno de los 5 dataset de train, y el dataset de test, en número de imágenes.

Dataset-	# img	Pistola	Smartphone	Billete	Monedero	Tarjeta	Background
1	4616	3464	0	0	0	0	1152
2	5412	3394	866	0	0	0	1152
3	5862	3394	866	134	137	179	1152
4	7177	3523	1022	287	315	307	1723
5	5801	1580	755	545	581	340	2000
Test	1100	294	115	123	104	64	400

Las predicciones, y las medidas precisión, recall y F1 obtenidas en cada clase por el modelo entrenado individualmente en la Dataset-1, -2, -3, -4 y -5 se muestran en la Tabla II. En general, el aumento del número de clases mejora el rendimiento global y por clase del modelo. De la Dataset-5 a Dataset-1, el rendimiento mejoró en 35.7 % en precisión, 8.7 % en recall y 28.05 % en F1. El clasificador reconoce la pistola más apropiadamente cuando aprende a distinguir mejor entre ella y más objetos diferentes como, smartphone, monedero, billete y tarjeta. Como se puede observar, los valores de precisión, recall y F1 obtenidos en las diferentes

[1] http://sci2s.ugr.es/weapons-detection

Tabla II
Para cada Dataset de train, predicciones, precisión, recall y F1 de cada clase, testeado en Dataset-Test.

Clasificador entrenado con Dataset-1		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	246	58	22	0	25	40	**62.91 %**	61.50 %	**62.19 %**
	Pistola	154	65	82	294	90	24	41.46 %	**100.00 %**	58.62 %
							MEDIA	52.19 %	80.75 %	60.41 %

Clasificador entrenado con Dataset-2		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	236	40	14	0	1	22	**75.39 %**	59.00 %	66.19 %
	Pistola	120	40	50	293	21	7	55.17 %	**99.65 %**	**71.03 %**
	Smartphone	44	43	40	1	93	35	36.32 %	80.86 %	50.13 %
							MEDIA	55.63 %	79.84 %	62.45 %

Clasificador entrenado con Dataset-3		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	228	1	6	0	2	3	**95.00 %**	57.00 %	71.25 %
	Billete	6	108	0	0	0	23	78.83 %	87.80 %	**83.07 %**
	Monedero	5	1	66	0	1	0	90.41 %	63.46 %	74.57 %
	Pistola	120	4	16	291	15	3	64.81 %	**98.97 %**	78.33 %
	Smartphone	39	8	15	3	95	6	57.22 %	82.60 %	67.61 %
	Tarjeta	2	1	1	0	2	29	82.85 %	45.31 %	58.58 %
							MEDIA	78.18 %	72.52 %	72.23 %

Clasificador entrenado con Dataset-4		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	302	5	14	2	6	4	90.69 %	75.50 %	82.40 %
	Billete	0	108	0	0	0	5	**95.57 %**	87.80 %	**91.52 %**
	Monedero	8	1	64	1	5	0	81.01 %	61.53 %	69.94 %
	Pistola	70	4	11	291	6	1	75.97 %	**98.97 %**	85.96 %
	Smartphone	18	3	15	0	97	5	70.28 %	84.34 %	76.67 %
	Tarjeta	2	2	0	0	1	49	90.74 %	76.56 %	83.05 %
							MEDIA	84.04 %	80.78 %	81.59 %

Clasificador entrenado con Dataset-5		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	338	3	7	1	1	1	**96.29 %**	84.50 %	90.01 %
	Billete	7	114	1	1	0	4	89.76 %	92.68 %	91.20 %
	Monedero	9	4	91	2	16	0	74.59 %	87.50 %	80.53 %
	Pistola	34	0	1	290	1	1	88.68 %	**98.63 %**	**93.39 %**
	Smartphone	8	1	2	0	97	1	88.99 %	84.34 %	86.60 %
	Tarjeta	4	1	2	0	0	57	89.06 %	89.06 %	89.06 %
							MEDIA	**87.89 %**	**89.45 %**	**88.46 %**

clases al entrenar el modelo en Dataset-1, -2 y -3 son muy desequilibrados, por ejemplo, para la clase smartphone, el modelo entrenado en Dataset-3 muestra una precisión del 57.22 %, recall 82.60 % y F1 67.61 % mientras que para la clase billete el modelo muestra un precisión de 78.83 %, recall 87.80 % y F1 83.07 %. Esto significa que el uso de un gran número de imágenes no implica un mejor aprendizaje sino que imágenes de mayor calidad proporcionan una mejora en el aprendizaje del modelo. El aprendizaje del modelo mejora sustancialmente desde Dataset-3 a Dataset-5 gracias a una mayor calidad y diversidad de las imágenes incluidas en Dataset-5. Los mejores resultados y más equilibrados por clase se obtienen al entrenar el modelo de seis clases, Dataset-5, ya que tiene una mayor calidad y diversidad. En el resto del trabajo utilizaremos Dataset-5 para la evaluación de OVA y OVO ya que es la base de datos que permite al modelo distinguir mejor entre los diferentes objetos.

IV. Evaluación de OVA, OVO y DRCW-OVO

En esta sección se evaluará el rendimiento de los métodos OVA, OVO y DRCW-OVO en nuestro problema de multi-clasificación. Para entrenar estos modelos, se ha usado la base de datos Dataset-5 y para testear las distintas técnicas y evaluarlas, Dataset-Test. Además, compararemos con el modelo entrenado sobre Dataset-5 al tener mejor resultados y ser el modelo de referencia. Primero analizamos OVA en IV-A, OVO en IV-B y DRCW-OVO en IV-C.

IV-A. Estrategia OVA

Para obtener los resultados del método de agregación MAX que vemos en la Tabla III, se han entrenado seis clasificadores y se han testeado sobre el conjunto de test, Dataset-Test. En este se pueden ver las predicciones, y las medidas precisión, recall y F1 obtenidas en cada clase.

Si comparamos los resultados de OVA con el modelo de referencia multiclase, observamos que los resultados son muy similares. El margen de mejora de OVA con respecto al modelo

Tabla III
PARA OVA MAX, PREDICCIONES, PRECISIÓN, RECALL Y F1 DE CADA CLASE, TESTEADO EN DATASET-TEST.

OVA MAX		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	335	4	8	1	1	0	**95.98 %**	83.75 %	89.45 %
	Billete	5	116	0	1	0	3	92.80 %	94.30 %	**93.54 %**
	Monedero	6	2	90	2	15	0	78.26 %	86.53 %	82.19 %
	Pistola	36	0	1	290	1	2	87.87 %	**98.63 %**	92.94 %
	Smartphone	13	1	3	0	97	1	84.34 %	84.34 %	84.34 %
	Tarjeta	5	0	2	0	1	58	87.87 %	90.62 %	89.23 %
							MEDIA	87.85 %	89.70 %	88.62 %

de referencia es insignificante. El modelo base entrenado en Dataset-5 obtuvo un precisión de 87,89 %, recall 89,45 % y un F1 88,46 %, mientras que OVA con el método MAX obtuvo un precisión de 87,85 %, recall 89,70 % y F1 88,62 %.

IV-B. Estrategia OVO

Nuestro problema es un problema de seis clases, por lo que OVO tendrá quince clasificadores. En la Tabla IV y para cada subtabla, se muestran los resultados de una estrategia de agregación diferente expresado en términos de predicciones, precisión, recall y F1 obtenidos para el test Dataset-Test.

Tabla IV
PARA CADA MÉTODO DE AGREGACIÓN DE OVO, PREDICCIONES, PRECISIÓN, RECALL Y F1 DE CADA CLASE, TESTEADO EN DATASET-TEST.

OVO VOTE random		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	337	3	7	1	1	1	**96.29 %**	84.02 %	89.87 %
	Billete	5	115	1	0	0	2	93.50 %	93.50 %	**93.50 %**
	Monedero	7	4	92	3	12	0	77.97 %	88.46 %	82.88 %
	Pistola	40	0	2	288	3	1	86.23 %	**97.96 %**	91.72 %
	Smartphone	8	0	2	2	98	2	87.50 %	85.22 %	86.34 %
	Tarjeta	3	1	0	0	1	58	92.06 %	90.62 %	91.34 %
							MEDIA	88.92 %	90.00 %	89.27 %

OVO VOTE by weight		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	341	4	8	1	1	1	**95.79 %**	85.25 %	90.21 %
	Billete	4	114	1	0	0	2	94.21 %	92.68 %	**93.44 %**
	Monedero	6	4	91	3	11	0	79.13 %	87.50 %	83.11 %
	Pistola	37	0	2	288	3	2	86.75 %	**97.96 %**	92.01 %
	Smartphone	10	0	2	2	99	1	86.84 %	86.09 %	86.46 %
	Tarjeta	2	1	0	0	1	58	93.55 %	90.62 %	92.06 %
							MEDIA	89.38 %	90.02 %	89.55 %

OVO WV		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	341	5	8	1	1	1	**95.52 %**	85.25 %	90.09 %
	Billete	4	114	0	0	0	2	95.00 %	92.68 %	**93.83 %**
	Monedero	6	3	92	3	11	0	80.00 %	88.46 %	84.02 %
	Pistola	37	0	2	288	3	2	86.75 %	**97.96 %**	92.01 %
	Smartphone	10	0	2	2	99	1	86.84 %	86.09 %	86.46 %
	Tarjeta	2	1	0	0	1	58	93.55 %	90.62 %	92.06 %
							MEDIA	89.61 %	90.18 %	89.75 %

OVO LVPC		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	342	5	9	2	1	2	94.74 %	85.50 %	89.88 %
	Billete	4	114	0	0	0	2	**95.00 %**	92.68 %	**93.83 %**
	Monedero	5	3	91	3	10	0	81.25 %	87.50 %	84.26 %
	Pistola	39	0	2	288	5	2	85.71 %	**97.96 %**	91.43 %
	Smartphone	8	0	2	1	98	1	89.09 %	85.22 %	87.11 %
	Tarjeta	2	1	0	0	1	57	93.44 %	89.06 %	91.20 %
							MEDIA	**89.87 %**	89.65 %	89.62 %

OVO ND		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	339	4	7	1	1	1	**96.03 %**	84.75 %	90.04 %
	Billete	4	114	1	0	0	2	94.21 %	92.68 %	**93.44 %**
	Monedero	6	4	90	3	11	0	78.95 %	86.54 %	82.57 %
	Pistola	40	0	2	289	3	2	86.01 %	**98.30 %**	91.75 %
	Smartphone	8	0	4	1	99	1	87.61 %	86.09 %	86.84 %
	Tarjeta	3	1	0	0	1	58	92.06 %	90.62 %	91.34 %
							MEDIA	89.15 %	89.83 %	89.33 %

OVO PC		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	341	5	8	1	1	1	95.52 %	85.25 %	90.09 %
	Billete	4	114	0	0	0	1	**95.80 %**	92.68 %	**94.21 %**
	Monedero	6	3	93	3	12	0	79.49 %	89.42 %	84.16 %
	Pistola	38	0	2	288	3	2	86.49 %	**97.96 %**	91.87 %
	Smartphone	9	0	1	2	98	1	88.29 %	85.22 %	86.73 %
	Tarjeta	2	1	0	0	1	59	93.65 %	92.19 %	92.91 %
							MEDIA	**89.87 %**	**90.45 %**	**90.00 %**

OVO PE		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	339	4	7	2	1	1	**95.76 %**	84.75 %	89.92 %
	Billete	4	114	0	0	0	2	95.00 %	92.68 %	**93.83 %**
	Monedero	6	4	91	3	11	0	79.13 %	87.50 %	83.11 %
	Pistola	40	0	2	288	3	2	85.97 %	**97.96 %**	91.57 %
	Smartphone	8	0	4	1	99	1	87.61 %	86.09 %	86.84 %
	Tarjeta	3	1	0	0	1	58	92.06 %	90.62 %	91.34 %
							MEDIA	89.26 %	89.93 %	89.43 %

Como podemos observar en la Tabla IV, OVO PC supera al resto de métodos, alcanzando unos valores medios de precisión del 89,87 %, recall 90,45 % y F1 90,00 %. Además, OVO PC obtuvo el mejor rendimiento sobre todas las estrategias evaluadas, OVA y el modelo multiclasificador base. En particular, OVO PC mejoró el modelo base en un 1,98 % en precisión, un 1 % en recall y un 1,54 % en F1.

IV-C. Estrategia DRCW-OVO

DRCW-OVO se evalúa en cuatro valores diferentes de k, el número de vecinos más cercanos que hacen el promedio medio en el valor de la distancia. Los resultados se muestran en la Tabla V. Cada subtabla muestra los resultados de un valor de k diferente expresado en términos de predicciones, precisión, recall y F1 obtenido sobre Dataset-Test.

Tabla V
PARA DISTINTOS VALORES DE K EN DRCW-OVO, PREDICCIONES, PRECISIÓN, RECALL Y F1 DE CADA CLASE, TESTEADO EN DATASET-TEST.

DRCW-OVO k=1		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	347	5	8	2	1	1	95.33 %	86.75 %	90.84 %
	Billete	2	114	0	0	0	1	**97.44 %**	92.68 %	**95.00 %**
	Monedero	5	3	91	3	9	0	81.98 %	87.50 %	84.65 %
	Pistola	35	0	2	288	4	1	87.27 %	**97.96 %**	92.31 %
	Smartphone	8	0	3	1	100	1	88.50 %	86.96 %	87.72 %
	Tarjeta	3	1	0	0	1	60	92.31 %	93.75 %	93.02 %
							MEDIA	**90.47 %**	**90.93 %**	**90.59 %**

DRCW-OVO k=3		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	343	5	9	2	1	1	95.01 %	85.75 %	90.14 %
	Billete	2	114	0	0	0	1	**97.44 %**	92.68 %	**95.00 %**
	Monedero	5	3	91	3	11	0	80.53 %	87.50 %	83.87 %
	Pistola	39	0	2	288	4	1	86.23 %	**97.96 %**	91.72 %
	Smartphone	8	0	2	1	98	1	89.09 %	85.22 %	87.11 %
	Tarjeta	3	1	0	0	1	60	92.31 %	93.75 %	93.02 %
							MEDIA	90.10 %	90.48 %	90.14 %

DRCW-OVO k=5		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	343	5	9	2	1	1	95.01 %	85.75 %	90.14 %
	Billete	2	114	0	0	0	1	**97.44 %**	92.68 %	**95.00 %**
	Monedero	5	3	91	3	10	0	81.25 %	87.50 %	84.26 %
	Pistola	39	0	2	288	5	1	85.97 %	**97.96 %**	91.57 %
	Smartphone	8	0	2	1	98	1	89.09 %	85.22 %	87.11 %
	Tarjeta	3	1	0	0	1	60	92.31 %	93.75 %	93.02 %
							MEDIA	90.18 %	90.48 %	90.19 %

DRCW-OVO k=7		Background	Billete	Monedero	Pistola	Smartphone	Tarjeta	Precisión	Recall	F1
Predicción	Background	341	5	9	2	1	1	94.99 %	85.25 %	89.86 %
	Billete	3	114	0	0	0	1	**96.61 %**	92.68 %	**94.61 %**
	Monedero	5	3	91	3	9	0	81.98 %	87.50 %	84.65 %
	Pistola	40	0	2	288	5	1	85.71 %	**97.96 %**	91.43 %
	Smartphone	8	0	2	1	99	1	89.19 %	86.09 %	87.61 %
	Tarjeta	3	1	0	0	1	60	92.31 %	93.75 %	93.02 %
							MEDIA	90.13 %	90.54 %	90.20 %

Como podemos observar en esta Tabla V, el método DRCW-OVO con valor k de uno, obtuvo el mejor resultado sobre todos los OVO evaluados, OVA y el modelo multiclasificador base. Esta configuración de DRCW-OVO ha obtenido un precisión media de 90.47 %, recall 90.93 % y F1 90.59 %. Esta técnica mejoró el modelo multiclasificador base en un 2,58 % en precisión, 1,48 % recall y 2,13 % F1. Esta mejora, con el alto valor de las medidas obtenidas, puede suponer un incremento importante en los resultados de la clasificación.

V. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo, proponemos el uso de técnicas de binarización como OVA y OVO para mejorar los resultados de una CNN normal en la tarea de clasificación. Por esta razón, y con la experiencia en este tipo de situaciones, utilizamos un conjunto de datos formado por armas de fuego y otros objetos que se manejan de forma similar. Los pasos realizados fueron, (1) incluir las clases de objetos que pueden suponer una confusión para el modelo como smartphone, billete, monedero y tarjeta, y añadir imágenes con más calidad y contexto a la base de datos utilizando distintas cámaras y contextos, (2) utilizar un nuevo modelo de clasificación como ResNet-101 y (3) mejorar la robustez utilizando técnicas de *machine learning* como OVA, OVO y DRCW-OVO.

Los resultados obtenidos del modelo entrenado en Dataset-5 y testeado sobre Dataset-Test obtienen una mejora de rendimiento de 35.7 % en precisión, 8.7 % recall y 28.05 % F1. Dataset-5 se usa para entrenar los modelos OVA, OVO y DRCW-OVO y los resultados del mejor modelo fueron de

90.47 % en precisión, recall de 90.93 % y F1 de 90.59 % para la técnica DRCW-OVO k=1. En resumen, hemos alcanzado una mejora del 2,58 % en precisión, del 1,48 % en recall y del 2,13 % en F1. Esto significa que, DRCW-OVO obtiene el valor más alto sobre todas las estrategias evaluadas.

Nuestro trabajo futuro consistirá en estudiar otros objetos que impliquen conflictos. Además, se hará un filtro para las instancias ruidosas que pueden causar confusión en los modelos CNN.

Agradecimientos

Este trabajo esta parcialmente respaldado por el Ministerio de Ciencia y Tecnología de España, en el proyecto TIN2017-89517-P y por la Junta de Andalucía en el proyecto P11-TIC-7765. Siham Tabik fue apoyada por el programa Ramon y Cajal (RYC-2015-18136) y Francisco Pérez por el Programa de personal técnico financiado por el programa operativo de empleo juvenil. Las Titan X Pascal usadas para esta investigación han sido donadas por NVIDIA Corporation.

Referencias

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[2] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[3] G. Flitton, T. P. Breckon, and N. Megherbi, "A comparison of 3d interest point descriptors with application to airport baggage object detection in complex ct imagery," *Pattern Recognition*, vol. 46, no. 9, pp. 2420–2436, 2013.

[4] A. Glowacz, M. Kmieć, and A. Dziech, "Visual detection of knives in security applications using active appearance models," *Multimedia Tools and Applications*, vol. 74, no. 12, pp. 4253–4267, 2015.

[5] R. K. Tiwari and G. K. Verma, "A computer vision based framework for visual gun detection using harris interest point detector," *Procedia Computer Science*, vol. 54, pp. 703–712, 2015.

[6] I. Uroukov and R. Speller, "A preliminary approach to intelligent x-ray imaging for baggage inspection at airports," *Signal Processing Research*, vol. 4, pp. 1–11, 2015.

[7] Z. Xiao, X. Lu, J. Yan, L. Wu, and L. Ren, "Automatic detection of concealed pistols using passive millimeter wave imaging," in *Imaging Systems and Techniques (IST), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1–4.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[9] R. Olmos, S. Tabik, and F. Herrera, "Automatic handgun detection alarm in videos using deep learning," *Neurocomputing*, 2017.

[10] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera, "An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes," *Pattern Recognition*, vol. 44, no. 8, pp. 1761–1776, 2011.

[11] M. Galar, A. Fernández, E. Barrenechea, and F. Herrera, "Drcw-ovo: distance-based relative competence weighting combination for one-vs-one strategy in multi-class problems," *Pattern recognition*, vol. 48, no. 1, pp. 28–42, 2015.

[12] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[14] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," in *Advances in neural information processing systems*, 2016, pp. 379–387.

[15] A. Rocha and S. K. Goldenstein, "Multiclass from binary: Expanding one-versus-all, one-versus-one and ecoc-based approaches," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 2, pp. 289–302, 2014.

[16] M. Yu, L. Gong, and S. Kollias, "Computer vision based fall detection by a convolutional neural network," in *Proceedings of the 19th ACM International Conference on Multimodal Interaction*. ACM, 2017, pp. 416–420.

[17] X. Chen, T. Fang, H. Huo, and D. Li, "Measuring the effectiveness of various features for thematic information extraction from very high resolution remote sensing imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 9, pp. 4837–4851, 2015.

[18] K. Öztürk and M. B. Yilmaz, "A comparison of classification approaches for deep face recognition," in *Computer Science and Engineering (UBMK), 2017 International Conference on*. IEEE, 2017, pp. 227–232.

[19] H. Lei and V. Govindaraju, "Half-against-half multi-class support vector machines," in *International Workshop on Multiple Classifier Systems*. Springer, 2005, pp. 156–164.

[20] P. Clark and R. Boswell, "Rule induction with cn2: Some recent improvements," in *European Working Session on Learning*. Springer, 1991, pp. 151–163.

[21] S. Knerr, L. Personnaz, and G. Dreyfus, "Single-layer learning revisited: a stepwise procedure for building and training a neural network," in *Neurocomputing*. Springer, 1990, pp. 41–50.

[22] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera, "Aggregation schemes for binarization techniques methods' description," *Pamplona, Spain*, 2011.

[23] J. Friedman, "Another approach to polychotomous classification," Technical report, Department of Statistics, Stanford University, Tech. Rep., 1996.

[24] E. Hüllermeier and K. Brinker, "Learning valued preference structures for solving classification problems," *Fuzzy Sets and Systems*, vol. 159, no. 18, pp. 2337–2352, 2008.

[25] J. C. Hühn and E. Hüllermeier, "Fr3: A fuzzy rule learner for inducing reliable classifiers," *IEEE Transactions on Fuzzy Systems*, vol. 17, no. 1, pp. 138–149, 2009.

[26] S. Orlovsky, "Decision-making with a fuzzy preference relation," in *Readings in Fuzzy Sets for Intelligent Systems*. Elsevier, 1993, pp. 717–723.

[27] A. Fernández, M. Calderón, E. Barrenechea, H. Bustince, and F. Herrera, "Enhancing fuzzy rule based systems in multi-classification using pairwise coupling with preference relations," *EUROFUSE*, vol. 9, pp. 39–46, 2009.

[28] T. Hastie and R. Tibshirani, "Classification by pairwise coupling," in *Advances in neural information processing systems*, 1998, pp. 507–513.

[29] T.-F. Wu, C.-J. Lin, and R. C. Weng, "Probability estimates for multi-class classification by pairwise coupling," *Journal of Machine Learning Research*, vol. 5, no. Aug, pp. 975–1005, 2004.

[30] O. Pele and M. Werman, "The quadratic-chi histogram distance family," in *European conference on computer vision*. Springer, 2010, pp. 749–762.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[32] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

Deep Learning for Fake News Classification

Miguel Molina-Solana
Data Science Institute
Imperial College London
London, UK
m.molina-solana@imperial.ac.uk

Julio Amador
Business School
Imperial College London
London, UK
j.amador@imperial.ac.uk

Juan Gómez Romero
Dept Computer Science and AI
Universidad de Granada
Granada, Spain
jgomez@decsai.ugr.es

***Abstract*—This work presents the application of several Deep Learning techniques for Natural Language Processing to the classification of tweets into containing fake news or not. To validate our approach, we use an open-access dataset containing annotated tweets related to the 2016 US elections. From our experiments, we can confirm that Deep Learning techniques are indeed able to identify tweets containing fake news, and that LSTMs with pre-computed embeddings is the best performing among the tested techniques (validation AUC = 0.70), particularly in avoiding misclassification of the minority class.**

***Index Terms*—deep learning, fake news, 2016 US elections**

I. Introduction

Since the 2016 US elections, the term 'fake news' has become mainstream and is nowadays commonly used to refer to pieces of information that are misleading, controversial or plainly inaccurate. This explosion has brought the attention of academics and practitioners from several fields, in an attempt to better understand the phenomenon and its causes.

Although the concept of fake news —as deliberately misleading pieces of information— is nothing new, the wide availability of social networking, publishing platforms, and general access to communication tools, has enabled their ultimate wide-spreading, overtaking the usual process of editorial curating.

Surprisingly enough, the acute characterization of fake news and its proper definition is elusive, remaining a fundamental question to be answered [11]. Cultural backgrounds, previous knowledge and ultimate interpretation of motives behind fake news play a prominent role on the interpretation of what it is and what it isn't a fake news, with plenty of academics sidelining the difficulty by focusing on the simpler issue of *false news* (to refer to those that have been fact-checked).

While diverse, the reasons for promoting fake news get often reduced to two [3]: pecuniary and ideological. Their impact, if widely embraced, believed and shared, can indeed be quite high as suggested by several pundits and academics in relation to the US presidential election [10].

With fake news becoming commonplace and being cheap to be generated, tools to flag controversial pieces of information are most welcome. Some approaches to identify fake news and their effects on behaviour have been suggested [8]. However, the question of how viral fake news effectively differ from other type of viral content remains unanswered.

Although focused on news stories and their mentions in tweets, a recent study from Vosoughi et al. [16] offers some insights as to how false news (as opposed to fact-checked verified news) might get spread. In particular, they report that falsehood diffused farther and faster than the truth despite structural elements of the network, not because of them.

On the other hand, Deep Learning models and techniques have demonstrated great performance and a very high potential in the recognition of complex patterns in several fields, especially in Computer Vision and Natural Language Processing. These models, which closely resemble the organization of neurons in the brain, mark the evolution of Neural Networks in an era of large data and very high computational power.

Neural Networks perform a non-linear transformation of the input values to the output values by means of several layers of interconnected computing units —i.e. the neurons. Their key approach is achieving learning by example: they take a (large) set of samples as training data, usually with already known labels, and automatically extract the relevant features that can be used to distinguish among classes, thus yielding a model able to classify new unknown samples. To do so, the training process applies an optimization algorithm to adjust the model parameters in order to minimize the network error.

In this context, it looks sensible to apply Deep Learning techniques to the task of classifying unseen pieces of information into containing fake news or not. The work reported here precisely looks into this, trying to offer some insights into the validity of this idea and its ultimate performance.

The paper is organized as follows. Next section introduces the dataset and algorithms we have used for the study. Section III presents the different experiments we carried out and their results and implications. The work finalizes with some pointers for further work.

II. Materials and methods

To test our hypothesis, we used a dataset containing tweets collected during 4 months just after the 2016 US presidential election [5], starting on November 8th. This dataset includes tweets that got re-tweeted more than 1000 times, and offer two sets of manual annotations for each tweet into fake news or not (Table I), attending to the categories established by [8]. For simplicity, the rest of this paper will consider a tweet as *fake* if it has been labeled as such by at least one annotator of the first or second team.

		2nd Label		
		Other Tweets	Fake News	Unknown
1st Label	Other Tweets	6482	1444	330
	Fake News	213	133	7
	Unknown	250	98	44

TABLE I
CONTINGENCY TABLE REPORTING THE DIFFERENCES AND THE SIMILARITIES BETWEEN THE LABELLING PERFORMED BY THE TWO TEAMS IN THE USED DATASET.

The complete dataset includes 9000 tweets and 20 variables, including the text of the tweet itself. After removing registers with empty values, we have 8336 tweets, with 6445 belonging to the negative class (*not fake*) and 1891 to the positive class (*fake*). Note that this procedure differs from the one reported by [4], which only takes into account the labelling by the second team.

We focus on the task of classifying tweets into fake news or not using tweet contents. The aim of this paper is to explore different approaches to build a classifier capable of tagging unseen viral tweets into fake news or not, and particular, those involving deep learning techniques. To do so, we will also limit ourselves to solely work with the actual textual content of the tweet.

Feed-forward Networks are the most basic and common type of neural networks. In these networks, computation moves in one direction, from the input to the output. The typical model is the perceptron, which defines several layers of fully-connected neurons. Model training is performed by backpropagation, a supervised learning algorithm in which the training loss, obtained as the difference between the expected result and the model output, is minimized after iterative adjustment of the model weight values. How the weights are adjusted is determined by an optimization algorithm such as the gradient descendent algorithm —and its variants for large data processing in Deep Learning; e.g. Stochastic Gradient Descendent (SGD), RMSProp, Adam, AdaGrad, etc.

Recurrent Neural Networks (RNNs) are a type of networks aimed at processing sequential data. Essentially, this type of networks compute a result not only from an input sample, but also by considering its previous state. Therefore, the model output for two equal samples can be different depending on the state of the network. Typically, RNNs process an input sequence x_{t-1}, x_t, ... to produce an output sequence o_{t-1}, o_t, ... Among them, long short-term memory (LSTM) networks [15] are particularly appropriate to deal with data sequences and time series with time lags of unknown size and duration between important events.

Formally, LSTM units are composed of a cell, an input gate, an output gate and a forget gate (see diagram in Figure 1). The cell is responsible for 'remembering' values over arbitrary time intervals —hence the word 'memory' in LSTM—, which are also passed towards the next units. Internally, the LSTM unit combines the input values, the values received through the input gate and the memory values to calculate the output values and the output gate values, taking as well into consideration the forget gate values.

Finally, and in order to validate our deep learning models built by following the LSTM approach, we compare their performance against traditional classification algorithms using only meta-data about each tweet and author; i.e. number of retweets, number of friends, etc. Specifically, we will use Random Forests [6], which are known to generally perform well in instance classification problems.

The software used for the experimentation is the R framework. We have used the caret package for creation and validation of the Random Forest classifiers [9] and the randomForest package for the underlying implementation of the algorithms [12]. For the deep learning techniques, we have used keras [1] with the tensorflow backend [2].

Due to our dataset being highly unbalanced (as shown in Table I), accuracy is not a proper metric to measure performance of the models. Instead we turn our attention to the ROC (receiver operating characteristic) curve and the AUC (area under the curve) measure, because they give a better overview of how true positive rate and false positive rate trade off.

III. Experiments and Discussion

A. Classification with metadata and Random Forest

In our first experiment, we have used the meta-data columns —all but the tweet text— to classify the tweets by the Random Forest algorithm. To do so, we have removed columns describing tweet id, user id, tweet creation date, and software used to publish the tweet. The ratio of training and validation datasets was set to 80-20%. After exploring a parameter grid through several experiments, parameter mtry, representing the number of variables randomly selected at each split of the forest generation process, was set to 4.

The results of this experiment can be seen in Figure 2, which shows the ROC curves of the classifiers trained with: (a) original data; (b) data augmented with the SMOTE algorithm [7] —using 200 and 100 as percentages for upsampling and downsampling, respectively. The threshold value for the class probability is automatically calculated by the randomForest package.

The AUC values can be seen in Table II. Results with the original data are unsatisfactory because of the strong bias of the classifier towards the majority class, and hence, the low sensitivity values. For this reason, although the AUC value of the classifier without augmentation is slightly better, we would select the classifier with augmentation as the baseline result to improve (AUC = 0.66, sensitivity = 0.74, specificity = 0.51).

The variable importance of the Random Forest classifier according to Gini measure the is shown in Figure 3. The results are consistent with the observations in [4] and [16]: the most descriptive variables are the number of retweets (i.e. the *virality* of the tweet) and the number of followers of the author. Note that other parameters, such as whether the user is verified or not, are not as relevant as they might be expected.

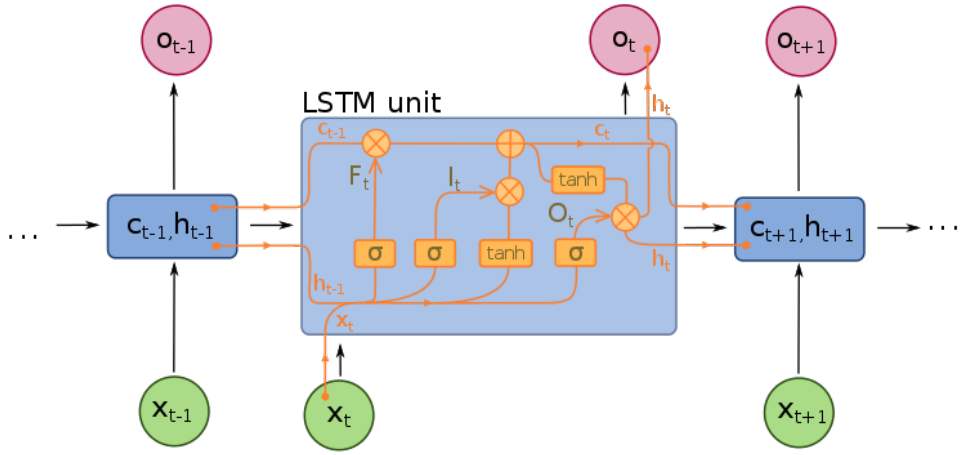


Fig. 1. Diagram of a LSTM unit

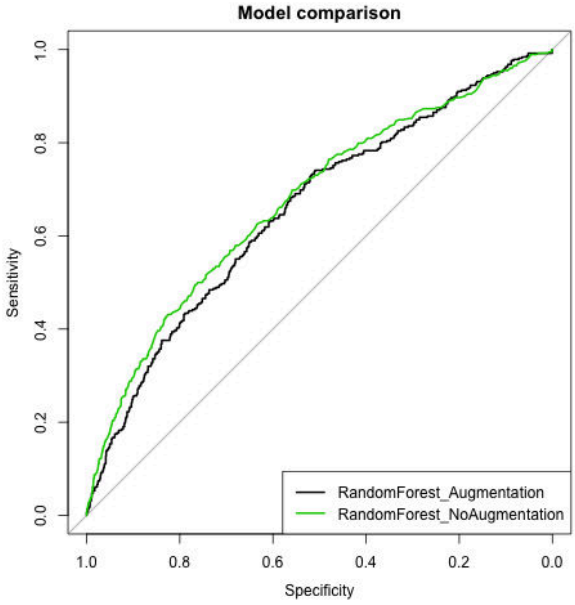


Fig. 2. ROC curves for the Random Forest classifiers

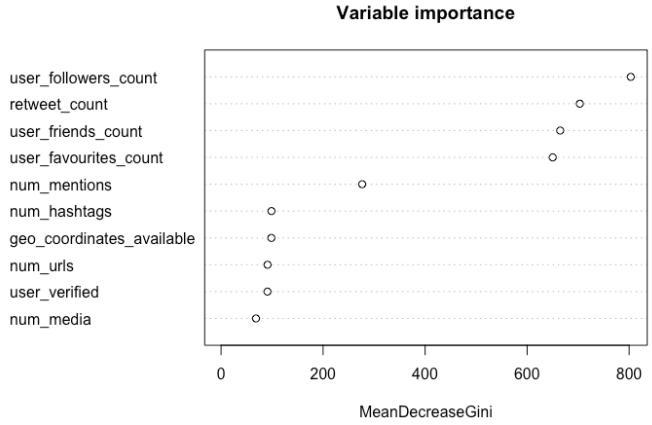


Fig. 3. Variable importance according to the Random Forest model with data augmentation

B. Classification with text and Deep Learning

In the second set of experiments, we have firstly used a feed-forward network to classify tweets from their contents. Secondly, we have implemented a LSTM network. In both cases, it is required to encode the input text into a numeric input, and then train the network to recognize if a tweet is a fake news or not.

There are several alternative approaches to perform the text-to-number encoding. A straightforward approach is one-hot encoding, in which texts are transformed into number vectors after the steps below:

1) Tokenize the tweets to extract all the words used in the texts.
2) Select the top-k used words ($k = 10,000$ in our case, from a total number of $21,421$ tokens).
3) Create a binary matrix where each column represents a top-k word and each row represents a tweet. Set the corresponding value $m_{i,j}$ to 1 if the word j appears in tweet i; otherwise, set it to 0.

This approach has several limitations, in particular because the sparse nature of the encodings resulting after step 3, which makes it difficult to train the classification model. Recently, it has been shown that it is better to encode texts as word embeddings. With this technique, we associate an n-dimension vector with each word, instead of a binary flag. Accordingly, a piece of text is a sequence of n-dimension vectors, being n typically 256, 512 or 1024. The translation from a word to a n-dimension vector is done in a way that the encoding somehow preserves the semantics of the original words; e.g. two encoding vectors corresponding to semantically related words will be close in the embeddings space.

This implies that embeddings must be also learnt in some way; e.g. by using algorithms such as the popular *word2vec* by Mikolov et al. [13]. While learnt word embeddings would be specific of the problem domain, it is common to use pre-

computed embeddings from large text corpora, thus avoiding the problem of reduced input data (as it happens in our problem) and long execution times (which require costly computational resources).

In this paper, we have used the Global Vectors for Word Representation (GloVe) version 1.2 [14]. From the publicly available editions, we have selected glove.6B, which is trained with Wikipedia 2014 and Gigaword 5 news dataset, and the 100-dimension embedding space.

After these considerations, we have performed the following experiments:

- Networks
  - Feed-forward networks (one embedding layer, two hidden layers and an output layer)
    - with self-trained embeddings
    - with GloVe word embeddings
  - LSTM networks (one embedding layer, a LSTM layer of 32 units and an output layer)
    - with self-trained embeddings
    - with GloVe word embeddings
- Tokenizer: default Keras text_tokenizer function
- Embeddings space size: 100
- Loss function: binary crossentropy
- Optimizer: RMSprop
- Epochs: < 10
- Batch size: 32
- Activation function for hidden layers: ReLU
- Activation function for output layer: sigmoid
- Training / validation: 80%, 20%

Note that, despite the simplicity of the network topologies, overfitting has frequently appeared in the experiments. In such cases, we have used the most accurate network obtained before detecting a significant decrease of the validation performance.

Figure 4 shows the ROC curves for validation of each one of the four models. It can be seen that the best model is the LSTM trained with GloVe word embeddings, which yields an AUC = 0.70 with sensitivity = 0.62 and specificity = 0.68 (see Table II for details).

C. Discussion

The main result from our work is that LSTM with pre-computed embeddings is the best performer among the techniques we have tested. It is particularly interesting that the ratio of false negatives, the main problem in an unbalanced dataset like ours, obtained with this technique is lower than in the other cases.

However, the overall improvement compared to a Random Forest using tweets' meta-data is not extremely high, as we can see in Figure 5. This poses the question as to how much is worth the extra computation incurred by LSTMs.

More interestingly, a more detailed study of the errors incurred by each model should be performed, in order to identify if both classifiers are orthogonal to some degree. If this is the case, the next step should be to build an ensemble method using these two classifiers, and to determine a proper aggregation function. From our initial studies, we anticipate that double-counting the positive detections (tweets including fake news) could improve the current results.

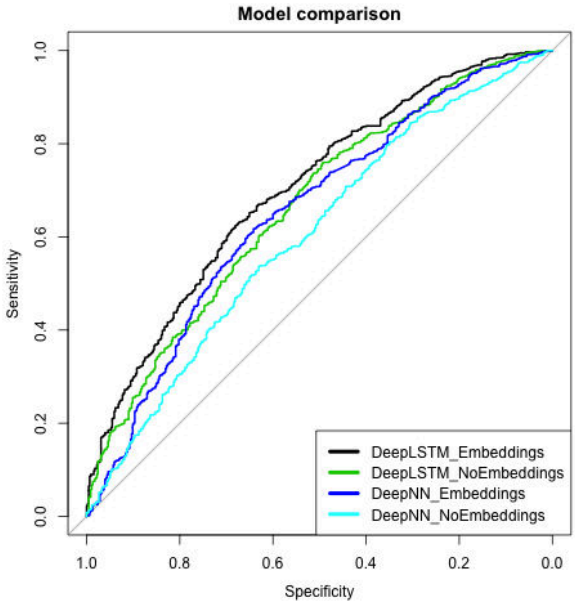


Fig. 4. ROC curves for the Deep Learning classifiers

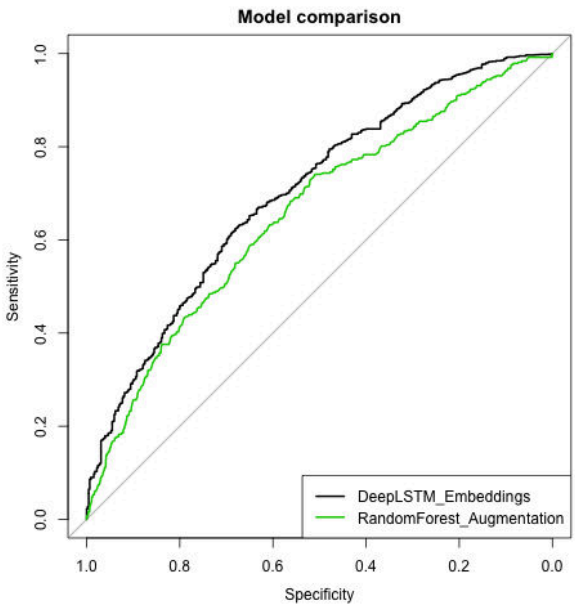


Fig. 5. ROC curves for the two best classifiers

IV. Future work

In this work, we have shown some preliminary results with off-the-shelf deep learning techniques applied to tweets' text in order to identify fake news. Even without a fine-grained adjustment of hyper-parameters nor a long training phase, results are quite promising over the tested dataset.

TABLE II
EXPERIMENT RESULTS

		Training				Validation			
Algorithm	Version	Accuracy	Sensitivity	Specificity	AUC	Accuracy	Sensitivity	Specificity	AUC
RandomForest	NoAugmentation	1.00	1.00	1.00	1.00	0.71	0.49	0.77	0.68
RandomForest	Augmentation	1.00	1.00	1.00	1.00	0.56	0.74	0.51	0.66
DeepNN	NoEmbeddings	0.99	1.00	0.99	1.00	0.56	0.54	0.63	0.60
DeepNN	Embeddings	0.96	0.98	0.95	1.00	0.62	0.62	0.64	0.65
DeepLSTM	NoEmbeddings	0.95	0.95	0.94	0.99	0.70	0.76	0.49	0.67
DeepLSTM	Embeddings	0.76	0.77	0.76	0.83	0.64	0.62	0.68	0.70

We plan applying more complex Deep Learning models (including a more extensive adjustment of parameters), and couple them together with a text pre-processing stage in order to better clean the text (e.g. currently, we are not considering text elements such as emojis and hyperlinks). It can be also helpful to tune the embeddings representation, either by exploring other pre-calculated transformation or by training on large corpora of only Twitter data.

Furthermore, there seems to be opportunity for ensemble methods that take into account both the meta-data of tweets and their actual text. Adding together the classification capabilities of Random Forests on the meta-data and LSTMs on the text appears as a very promising line of action, which we are currently exploring.

Finally, this study (and hence its results) is tightly coupled to and heavily dependent on the employed dataset. A replication of the experiments on a complementary one is necessary to further validate the current findings strengthening the conclusions and eventually leading to further insights.

All in all, the application of Deep Learning techniques to the task of identifying fake news in social media looks like a very relevant and timely application.

ACKNOWLEDGEMENTS

The authors want to thank the support of the Data Science Institute, Imperial College London. M. Molina-Solana is receiving funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 743623. Juan Gómez-Romero is supported by University of Granada under the Young Researchers Fellowship, and received funding from the Spanish Ministry of Education, Culture and Sport under the José Castillejo Research Stays Programme.

REFERENCES

[1] J. Allaire and F. Chollet. *keras: R Interface to 'Keras'*. R package version 2.1.6.9001.

[2] J. Allaire and Y. Tang. *tensorflow: R Interface to 'TensorFlow'*. R package version 1.5.0.9001.

[3] H. Allcott and M. Gentzkow. Social media and fake news in the 2016 election. Technical Report 23089, National Bureau of Economic Research, 2017.

[4] J. Amador, A. Oehmichen, and M. Molina-Solana. Characterizing Political Fake News in Twitter by its Meta-Data. *arXiv*, 2017.

[5] J. Amador, A. Oehmichen, and M. Molina-Solana. Fakenews on 2016 US elections viral tweets (November 2016 - March 2017). http://dx.doi.org/10.5281/zenodo.1048826, 2017.

[6] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[8] N. J. Conroy, Y. Chen, and V. L. Rubin. Automatic deception detection: Methods for finding fake news. In *Proceedings of the 78th ASIS&T Annual Meeting: Information Science with Impact: Research in and for the Community*, pages 82:1–82:4, 2015.

[9] M. K. C. from Jed Wing, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, Z. Mayer, B. Kenkel, the R Core Team, M. Benesty, R. Lescarbeau, A. Ziem, L. Scrucca, Y. Tang, C. Candan, and T. Hunt. *caret: Classification and Regression Training*, 2018. R package version 6.0-79.

[10] R. Gunther, E. C. Nisbet, and P. Beck. Fake news may have contributed to trump's 2016 victory. Technical report, Ohio State University, 2018.

[11] D. M. J. Lazer, M. A. Baum, Y. Benkler, A. J. Berinsky, K. M. Greenhill, F. Menczer, M. J. Metzger, B. Nyhan, G. Pennycook, D. Rothschild, M. Schudson, S. A. Sloman, C. R. Sunstein, E. A. Thorson, D. J. Watts, and J. L. Zittrain. The science of fake news. *Science*, 359(6380):1094–1096, 2018.

[12] A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[14] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[15] S. H. J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[16] S. Vosoughi, D. Roy, and S. Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018.

Procesamiento, análisis y clasificación de neuroimagen con arquitecturas Deep Learning híbridas

Andrés Ortiz[a*], Francisco J. Martinez-Murcia[b†], Jorge Munilla[a‡], Juan M. Górriz[b§], Javier Ramírez[b¶] and Fermín Segovia[b‖]
[a]*dept. of Communications Engineering*, *University of Málaga*. Málaga, Spain.
Emails: *aortiz@ic.uma.es, ‡munilla@ic.uma.es
[b]*Dept. of Signal Theory,Networking and Communications*, *University of Granada*, Granada, Spain
Emails: †fjesusmartinez@ugr.es, §gorriz@ugr.es, ¶javierrp@ugr.es, ‖fsegovia@ugr.es

***Abstract*—En este trabajo presentamos los últimos trabajos realizados conjuntamente por los grupos SiPBA y BioSiP, donde se han desarrollado diferentes métodos basados en aprendizaje estadístico y Deep Learning para el procesamiento de imágenes cerebrales en diferentes modalidades. Los métodos desarrollados se han aplicado al diagnóstico precoz de enfermedades neurodegenerativas, como la enfermedad de Alzheimer y la enfermedad de Parkinson, proporcionando diferentes representaciones de las imágenes y mejorando las tasas de clasificacón de otros sistemas CAD previamente publicados.**

***Index Terms*—Deep Learning, Computer aided diagnosis, Alzheimer, Parkinson, statistical processing.**

I. Introducción

Los nuevos métodos basados en aprendizaje estadístico e inteligencia artificial en combinación con las arquitecturas masivamente paralelas disponibles en la actualidad (unidades de procesamiento gráfico, GPU, por ejemplo) han supuesto una revolución en la forma de abordar el procesamiento de imágenes médicas y en los sistemas de diagnóstico automático. En este trabajo se presentan los últimos métodos desarrollados en los grupos de investigación SipBA y BioSiP en este sentido, aplicados al procesamiento de imágenes cerebrales estructurales (resonancia magnética, IRM) y funcionales (Tomografía por emisión de positrones, PET o tomografía por emisión de un único fotón, SPECT con diferentes radiofármacos) para el diagnóstico de enfermedades neurológicas y neurodegenerativas como la enfermedad de Alzheimer y Parkinson. En el primer trabajo [1], se muestra un sistema que hibrida el aprendizaje profundo (Deep Learning, DL) con máquinas de soporte vectorial para combinar datos de imagen IRM y PET. En el segundo trabajo [2], se presenta un método para extraer características de una imágen tridimensional a partir de su descomposicion en componentes empíricas, utilizando curvas fractales. En el tercer trabajo [3], se muestra una arquitectura que combina varias redes profundas siamesas, para el diagnóstico del Parkinson y finalmente, en [4], se presenta un estudio detallado sobre influencia del preprocesamiento de las imágenes en las redes convolucionales profundas para el diagnóstico del Parkinson.

En los trabajos mencionados se han utilizado imágenes de repositorios internacionales ampliamente utilizados por investigadores de diferentes ámbitos, como el *Alzheimer's Disease Neuroimaging Initiative, ADNI* para Alzheimer y *Parkinson Progression Markers Initiative, PPMI* para Parkinson.

II. Ensembles de arquitecturas profundas para el diagnóstico del Alzheimer

La enfermedad de Alzheimer afecta progresivamente tanto a la estructura del cerebro como a la actividad funcional. La evaluación de ambas requiere modalidades diferentes: en el primer caso, se utilizan imágenes de resonancia magnética y en el segundo, imágenes PET con fluorodeoxiglucosa como radiofármaco. Dado que las imágenes utilizadas son de una resolucion considerable y en 3D, el procesamiento de las imágenes completas como es común en 2D no es posible en este caso, debido, fundamentalmente, a los requisitos de memoria. Por otro lado, extraer patrones de imágenes con gran número de voxeles, requiere de arquitecturas con un gran número de neuronas y por tanto, muy propensas al sobreentrenamiento. Para solucionar estas dos cuestiones, en [1] se propone definir clasificadores débiles *Deep Belief Networks (DBN)*, sobre cada región neuroanatómica, de acuerdo al atlas AAL. Finalmente, se combinan las predicciones individuales. Se proponen, además, dierentes estrategias de combinación (*ensemble*) de las redes, desde una simple votación por mayoría hasta unaa votación con pesos calculados a partir de la significancia estadística de los voxeles de cada región en el diagnóstico diferencial. También se propone una arquitectura híbrida donde los pesos para la combinación de las predicciones parciales se calculan mediante un el proceso de optimización de un SVM.

III. Empirical Functional PCA for 3D image feature extraction through fractal sampling. Application to PET image classification

La idea principal de este trabajo consiste en expresar una imagen de prueba como combinación lineal de un conjunto de imágenes de entrenamiento. Dicho de otra forma, se trata de

representar una imagen en una base formada por un conjunto que contenga imágenes de ambas clases control y Alzheimer). A diferencia de alternativas que utilizan las imágenes completas para determinar una base [6], [8], en este trabajo se utiliza la descomposición empírica de modos (EMD) para obtener una base con una mayor capacidad de representación. La descomposición empírica de modos proporciona un medio para descomponer una imágen en componentes de diferente frecuencia y que por tanto, contienen diferente información. Una descomposición similar puede hacerse mediante análisis de Fourier o de Wavelet. No obstante, ambos métodos utilizan una base predefinida para representar la señal original, mientras que EMD calcula dicha base de forma empírica para la señal a descomponer. Aunque existen extensiones de EMD a 2D que hemos aplicado anteriormente con el mismo propósito que en este trabajo [7], la extensión a 3D no es inmediata y computacionalmente muy costosa. En este trabajo proponemos convertir la imagen 3D en 1D utilizando curvas fractales multidimensionales para muestrear los vóxeles de la imagen. Concretamente curvas de Peano-Hilbert, que tienen la propiedad de conservar la relación espacial entre los puntos de la curva, asegurando que puntos próximos en 3D estarán también próximos en 1D. Una vez descompuesta la señal, se utiliza el algoritmo *Basis Pursuit* para encontrar la mejor representación de la misma en la base sobrecompleta formada por las funciones de modo intrínseco (IMF). Dichas representaciones son finalmente utilizadas para entrenar un SVM.

IV. Label Aided Deep Ranking for the Automatic Diagnosis of Parkinsonian Syndromes

En este trabajo se extiende el método Deep Ranking (DR) con el fin de aplicarlo eficientemente a la clasificación de imágenes DaTSCAN SPECT, y por tanto, al diagnóstico diferencial de la enfermedad de Pakinson. La idea principal de DR es obtener una función que permita proyectar las imágenes 3D en un espacio de dimensión $\mathbb{R}^n$. Dicha función se obtiene mediante un proceso de aprendizaje que utiliza una función de pérdida cuya minimización provoca que muestras de la misma clase estén juntas en el espacio de proyección, mientras que muestras de clases diferentes estén lo más separadas posible en dicho espacio, utilizando como métrica la distancia Euclídea. El modelo DR desarrollado, está formado por tres redes siamesas (que comparten la actualización de los pesos). De esta forma, tras la convergencia, las tres redes se comportarán de la misma forma (dado que comparten los pesos optimizados durante el aprendizaje). En este trabajo se han utilizado redes convolucionales 3D en cada bloque que contienen 5 capas de convolución y dos capas *Fully connected* así como regularización mediante *Dropout*.

V. Convolutional Neural Networks for Neuroimaging in Parkinson's disease: is preprocessing needed?

En el campo de la neuroimagen, es una práctica común normalizar las imágenes tanto espacialmente como en intensidad. La normalización espacial se realiza de acuerdo a una plantilla, a través de transformaciones afines para realizar la corrección del movimiento y la reorientación de las imágenes. De esta forma, los voxeles de todas las imágenes coinciden, refiriéndose a la misma posición neuroanatómica. La normalización en intensidad resulta especialmente útil con imágenes funcionales, donde la mayor parte de la información está en el nivel de activación de los voxeles. En este trabajo, se realiza un estudio detallado de la influencia de ambas normalizaciones en la clasificación de imágenes DaTSCAN SPECT utilizando redes convolucionales 3D (CNN). Se muestran los resultados obtenidos para diferentes artuitecturas y funciones de activación y coste, analizando también la dispersión de los resultados obtenidos con validación cruzada. Del estudio realizado se concluye que los clasificadores basados en CNN permiten obtener un área bajo la curva ROC muy alta (0.98) sin normalizar espacialmente las imágenes, de forma que es posible evitar este costoso proceso. Sin embargo, la normalización en intensidad sí influye de forma decisiva en las prestaciones del modelo. Por otro lado, la visualización de los patrones de déficit dopaminérgico descubiertos a través de los mapas de saliencia, coinciden con los que aparecen en la literatura para imágenes FP-CIT, validando así la utilidad de la metodología propuesta.

Acknowledgment

Este trabajo ha sido parcialmente financiado por MINECO/FEDER bajo los proyectos TEC2015-64718-R and PSI2015-65848-R projects.

References

[1] Ortiz, A., Munilla, J., Górriz, J.M. and Ramírez, J., "Ensembles of Deep Learning Architectures for the early diagnosis of Alzheimer's Disease". International Journal of Neural Systems. 26(7):1-23. 2016.

[2] Ortiz, A., Munilla, J., Martínez-Murcia, F., Górriz, J.M. and Ramírez, J.,"Empirical Functional PCA for 3D image feature extraction through fractal sampling". International Journal of Neural Systems, 2018. Accepted for Publication

[3] Ortiz, A., Martinez-Murcia F.J., Munilla, J., Górriz J.M. and Ramírez J., "Label Aided Deep Ranking for the automatic diagnosis of Parkinsonian Syndromes.". 2018, submitted.

[4] Martínez-Murcia, F., Górriz, J.M., J. Ramírez and Ortiz, A. "Convolutional Neural Networks for NEuroimaging in Parkinson's disease: is preprocessing needed?". International Journal of Neural Systems, 2018. Accepted for publication

[5] Ortiz, A., Lozano, F., Górriz, J.M., Ramírez, J., and Martínez-Murcia, F.J., "Discriminative Sparse Features for Alzheimer's Disease Diagnosis using multimodal image data". Current Alzheimer Research. 2018;15(1):67-79

[6] Álvarez, I., Górriz, J.M., Ramírez, J., Salas-Gonzalez, D., López, M., Puntonet, G.C. and Segovia, F. "Alzheimer's diagnosis using eigenbrains and support vector machines". Electronics Letters, 2009; 45(7):342-343.

[7] Rojas, A., Górriz, J.M., Ramírez, J., Illán, A., Martínez-Murcia, F.J., Ortiz, A., and Gómez-Rio, M., "Application of Empirical Mode Decomposition (EMD) on DaTSCAN SPECT images to explore Parkinson Disease". Expert Systems with Applications. 2013 40(7):2756-2766.

[8] J.M. Górriz; J. Ramírez; J. Suckling; I.A. Illán; A. Ortiz; F.J. Martinez-Murcia; F. Segovia; D. Salas-González and S. Wang. "Case-based statistical learning: a non parametric implementation with a conditional-error rate SVM". IEEE Access, 2017(5):11468-11478.

I Workshop en Deep Learning (DeepL)

SESIÓN 3

Guiding the Creation of Deep Learning-based Object Detectors*

Ángela Casado
Departamento de Matemáticas y Computación
Universidad de La Rioja
Logroño, España
acg181293@hotmail.com

Jónathan Heras
Departamento de Matemáticas y Computación
Universidad de La Rioja
Logroño, España
jonathan.heras@unirioja.es

***Abstract*—Object detection is a computer vision field that has applications in several contexts ranging from biomedicine and agriculture to security. In the last years, several deep learning techniques have greatly improved object detection models. Among those techniques, we can highlight the YOLO approach, that allows the construction of accurate models that can be employed in real-time applications. However, as most deep learning techniques, YOLO has a steep learning curve and creating models using this approach might be challenging for non-expert users. In this work, we tackle this problem by constructing a suite of Jupyter notebooks that democratizes the construction of object detection models using YOLO. The suitability of our approach has been proven with a dataset of stomata images where we have achieved a mAP of 90.91%.**

***Index Terms*—Object Detection, YOLO, Jupyter Notebooks.**

I. Introduction

Object detection is a computer vision area that focuses on identifying the position of multiple objects in an image. Traditionally, object detection methods have been based on two main techniques known as sliding windows and image pyramids; and, they have been successfully applied to solve problems such as face detection [1] or pedestrian detection [2]. However, those approaches are slow, lack the notion of aspect ratio and are error prone [3]. These problems have been recently overcome using deep learning techniques.

Deep learning has impacted almost every area of computer vision, and object detection is no exception. Intuitively, in deep learning-based object detectors, we input an image to a network and obtain, as output, the bounding boxes (that is, the minimum rectangle containing the objects) and the class labels. Deep learning-based object detectors can be split into two groups: one-stage and two-stage object detectors. The former divide the image into regions, that are passed into a convolutional neural network, and then the prediction is obtained — these detectors include techniques such as SSD [4] or YOLO [5]. The two-stage object detectors employ region proposal methods to obtain interesting regions in the image, that are later processed to obtain the prediction — these methods include the R-CNN family of object detectors [6]–[8].

*This work was partially supported by Ministerio de Economía y Competitividad [MTM2017-88804-P]. We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

Usually, one-stage object detectors are faster but less precise than two-stage object detectors; however, the one-stage object detector YOLO has recently achieved a similar accuracy to the one obtained by two-stage object detectors, but keeping fast processing [5]. Due to this fact, the YOLO object detector has been applied in problems that require real time processing but also need a high accuracy, like real time detection of lung modules [9] or the detection of small objects in satellite imagery [10].

The YOLO object detector is provided as part of the Darknet framework [11]; but, as almost every deep learning tool, it has a steep learning curve and adapting it to work in a particular problem might be a challenge for several users — apart from understanding the underlying algorithm of YOLO (a step that might not be necessary to create an object detection model with YOLO), using YOLO might be a challenge since it requires, among others, the installation of several libraries, the modification of several files and the usage of a concrete file structure. Therefore, even if this technique might be helpful for several fields (ranging from biomedicine and agriculture to security), users from those fields are not able to take advantage of it. To solve this problem, we have developed an assistant, in the form of a suite of Jupyter notebooks, that guides non-expert users through all the steps that are necessary in an object detection project using YOLO.

As we explain in Section II, there are several steps that are required to create a deep learning-based object detector; but, thanks to our assistant, see Section III, they are reduced to fixing a few parameters, and the rest of the process is conducted automatically by our tool. To prove the suitability of our approach, we have employed the assistant to create an stomata detector for plant images, see Section IV. The paper ends with some conclusions and further work.

II. A Pipeline to create deep learning-based object detectors

In this section, we present the common workflow to create a deep learning-based object detector (see Figure 1), the challenges that are faced on each stage of the pipeline, the solutions that we propose to deal with those challenges, and the particularities of using the YOLO network in each stage.

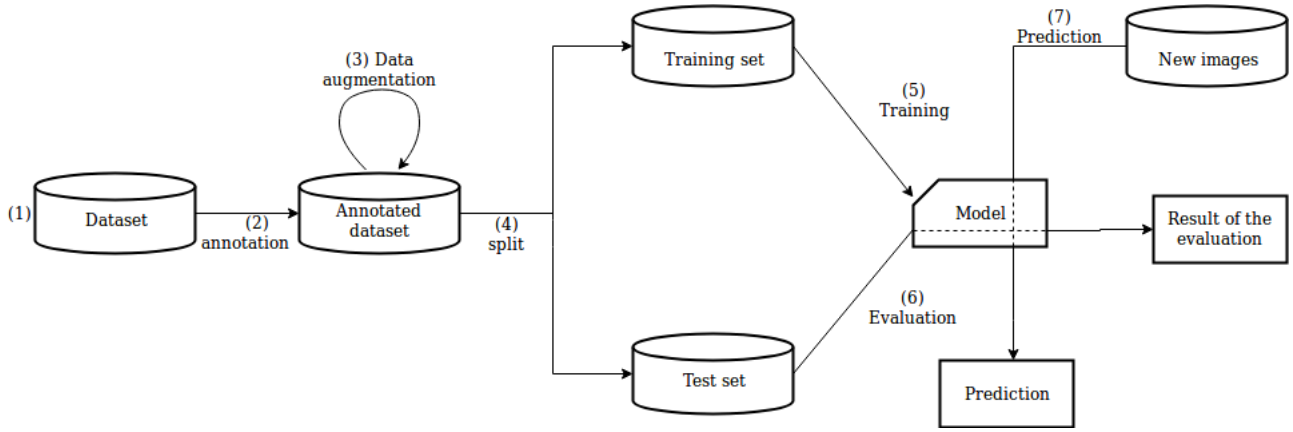


Fig. 1. Workflow of object detection projects

1. Dataset acquisition

Independently of the framework employed to create an object detection model, the first step to construct an object detection model is always the acquisition of a dataset of images containing the objects that we want to detect. This task is far from trivial since the dataset must be representative of what the model will find when deployed in the real world; and, therefore, its creation must be carefully undertaken. Moreover, acquiring data in problems related to, for instance, biomedical images might be difficult [12], [13].

There are several ways of obtaining a dataset of images depending on the project, but we can identify four main sources:

- *Open datasets*. There are several projects that have collected a huge amount of images, such as ImageNet [14] or Pascal VOC [15]; however, those datasets might not contain the objects that the user is interested in detecting.
- *Web scraping*. It is quite straightforward to create a program that uses image search engines, such as Google images or Bing images, to download images with a Creative Commons License from the web — this allows us to legally employ those images to create object detectors. Even if this approach can be a fast way of downloading images, it might require data curation to obtain a representative dataset for the intended task.
- *Special purpose datasets*. This approach might be the only option in problems with sensitive data, like in biomedicine, where an external agent (for instant, a hospital) provides the images. The main drawbacks for this approach are the limited number of images, the difficulties to acquire new data, and restrictions related to access and use of the images.
- *Special purpose devices to capture images*. In this case, the images are acquired by using devices that might range from a microscope to a fixed camera in a working environment. This approach has the advantage of dealing with images that are representative of the environment where the model will be later used; and, additionally, it is usually easy to obtain new images. However, the models created with those images might not be useful if the conditions change. For instance, if a model is created using images acquired with a microscope using a set of fixed conditions, the model might not work as expected if applied to images acquired with a different microscope or under different conditions.

As we have previously mentioned, this step is independent of the tool that is employed to create the object detection model. The only restriction from the YOLO side is that the images must be stored using the JPG format.

2. Dataset annotation

Once the images have been acquired, they must be annotated, a task that is time-consuming and might require experts in the field to conduct it properly [16]. In the object detection context, images are annotated by providing a file with the list of bounding boxes and the category of the objects inside those boxes. Those annotation files are not written manually, but they are built using graphical tools like LabelImg [17] or YOLO mark [18]. Unfortunately, there is not a standard annotation format and, in fact, the format varies among object detection frameworks, and also among the annotation tools. Therefore, this must be taken into account when annotating a dataset of images; otherwise, a conversion step will be necessary after the annotation process is completed.

In the case of YOLO, the annotations are provided as plain text files where each bounding box is given by the position of its center, its width and height. It is convenient to use a tool that produces the annotations directly in this format, like YOLO mark [18], but there are also converters from other formats; for instance, from Pascal VOC or Kitti [19].

3. Dataset augmentation

As we have previously mentioned, acquiring and annotating datasets of images for object detection problems might be a challenge; this might lead to generalization problems if enough images are not acquired. A successful method that has been applied to deal with the issue of limited amount of data is *data augmentation* [20]. This technique consists in generating

new training samples from the original dataset by applying image transformations (for instance, applying flips, rotations, filters or adding noise). This approach has been applied in image classification problems, and there are several libraries implementing this method (for instance, Augmentor [21] or Imgaug [22]).

The application of data augmentation is not straightforward in the case of object detection due to the fact that, on the contrary to data augmentation in image classification, transformation techniques alter the annotation in the context of object detection. For instance, applying the vertical flip operation to a cat image produces a new cat image — i.e. the class of the image remains unchanged — but the position of the cat in the new image has changed from the original image. Therefore, we have to transform not only the image but also the annotation.

In order to apply data augmentation in object detection, the usual approach has consisted in implementing special purpose methods, depending on the particular problem, or manually annotating artificially generated images. Neither of these two solutions is feasible when dealing with hundreds or thousands of images. To deal with this issue, we can employ the CLoDSA library [23], a tool that can be applied to automatically augment a dataset of images devoted to classification, localization, detection or semantic segmentation using a great variety of the classical image augmentation transformations. Using this tool, it is possible to automatically generate a considerable amount of images, together with their annotations, starting from a small dataset of annotated images. CLoDSA is compatible with the YOLO format.

4. Dataset split

As in any other kind of machine learning project, it is instrumental to split the dataset obtained in the previous steps into two independent sets: a training set — that will be employed to train the object detector — and a test set — that will be employed to evaluate the model. Common split sizes for training and testing set include 66.6%/33.3%, 75%/25%, and 90%/10%, respectively.

In the case of YOLO, the dataset split is achieved by using a particular folder structure to store the images and the labels that will be employed for training and testing, and providing two files that indicate, respectively, the set of images that will be employed for training and testing. It is worth mentioning that the YOLO framework is really sensitive to such a structure, and small changes will prevent the user from training the model.

5. Training the model

Given the training set of images, several tasks remain before starting the process to train an object-detection model. In particular, it is necessary to define the architecture of the model (that is, the number and kind of layers), fix some hyper-parameters (for example, the batch size, the number of epochs, or the momentum) and decide whether the training process starts from scratch or some pre-trained weights are employed — the latter option, known as fine-tuning, usually improves the training process [24].

The YOLO framework supplies several pre-defined models for training an object detector — the best model, both in terms of accuracy and time efficiency, is the YOLO model v3. The architecture and the hyper-parameters of those models are defined in a configuration file; and, even if the by-default YOLO hyper-parameters usually work properly for training a new model, it is necessary to modify the configuration files since the model architecture must be adapted depending on the number of classes included in the dataset. Once the configuration files have been adapted, the training process can start just by executing a command. The process will end either after the provided number of epochs is reached or when the user decides to manually stop the process. In order to train a YOLO model, it is recommended to use some pre-trained weights that must be downloaded and included in the project [5].

6. Evaluating the model

After training the model, we need to evaluate it to assess its performance. Namely, for each of the images in the testing set, we present it to the model and ask it to detect the objects in the image. Those detections are compared to the ground-truth provided by the annotations of the testing set, and the result of the comparison is evaluated using metrics such as the IoU, the mAP, the precision, the recall or the F1-score [15]. The YOLO framework can perform such an evaluation automatically provided that several configuration files are correctly defined, and the correct instruction is invoked.

A problem that might arise during the evaluation is the overfitting of the object detection model; that is, the model can detect objects on images from the training dataset, but it cannot detect objects on any others images. In order to avoid this problem, *early stopping* [25] can be applied by comparing the results obtained by the models after different numbers of epochs. In the case of YOLO models, early stopping can be applied since the framework saves the state of the model every time that a certain number of training iterations is reached.

7. Deploying the model

Finally, the model is ready to be employed in images that neither belong to the training set nor to the testing set. Even if using the model with new images is usually as simple as invoking a command with the path of the image (and, probably, some additional parameters), it is worth mentioning that it is unlikely that the person who created the object detection model is also the final user of such a model. Therefore, it is important to create simple and intuitive interfaces that might be employed by different kinds of users; otherwise, they will not be able to take advantage of the object detection model.

III. A suite of Jupyter notebooks

As we have explained in the previous section, training and using a YOLO model involves the creation and modification of several configuration files, the use of a concrete folder

structure, and the execution of several commands; hence, the process is time-consuming and error prone. In this section, we introduce a simple-to-use tool that facilitates the creation of YOLO-based object detectors using Jupyter notebooks — the suite of notebooks can be downloaded from https://github.com/ancasag/YOLONotebooks.

Jupyter notebooks [26] are documents for publishing code, results and explanations in a form that is both readable and executable. Jupyter notebooks have been widely adopted across multiple disciplines, both for its usefulness in keeping a record of data analyses, and also for allowing reproducibility. In addition, Jupyter notebooks are a useful tool to teach and learn concepts from data science and artificial intelligence [27], [28]. In our case, we have developed a suite of notebooks that guides the user in the process to create a YOLO-based object detector. The only requirement to run the notebooks is the installation of the programming language Python and its Jupyter library.

The suite of notebooks is open-source and contains several notebooks that introduce several traditional object detection notions, explain how to install the YOLO framework, and provide several examples showing how to use it (for instance, showing how the pre-trained models included in the framework might be used to detect objects in images and videos). However, the most important notebook of the suite is the one that automates the process of creating a new object detection model. Using this notebook, the user only has to (1) create a folder containing the images and the annotations in the YOLO format, and (2) fix 4 parameters in the notebook (the name of the project, the path to the folder containing the images and annotations, the list of classes, and the percentage of the images that will be employed for training); the rest of the process is carried out by simply following the steps included in the notebook. In particular, the notebook is in charge of:

- Validating that the images of the dataset are given in the correct format.
- Checking that all the images of the dataset have their corresponding annotation.
- Guiding the user in the process to augment the dataset of images using the CLoDSA library.
- Splitting the dataset into a training set and testing set, and organizing those sets in the way required by the YOLO framework.
- Creating all the configuration files for training, evaluating and deploying the model using the last version of the YOLO network.
- Generating all the instructions that are required to train, test and deploy the YOLO model.
- Providing a simple way of invoking the generated YOLO model to detect objects in new images.

Without our tool, all those steps should be carried out manually; hence, the burden of creating a YOLO-based object detector is significantly reduced. The aforementioned functionality has been implemented in Python using several third-party libraries such as Scikit-learn [29] and OpenCV [30].

IV. Case study: stomata detection

In this section, we show the feasibility of using our tool by creating a stomata detector in images of plant leaves. A stoma is a tiny opening, or pore, that is used for gas exchange in plants. The amount and behavior of stomata provide key information about water stress levels, production ratio, and, in general, the overall health of the plant [31]. Hence, by measuring the number of stomata, it is possible to manage better the resources in agriculture and obtain better yields [32]. However, manually counting the number of stomata is a time-consuming and subjective task due to the considerable amount of stomata in an image, their small size, and the fact that it is necessary to analyze batches of dozens of images. Therefore, it is useful to automatically detect and count the number of stomata in plant images.

Several studies can be found on automatic detection of stomata, but they employ traditional features like Haar [33], MSER [34] or HOG [35] combined with a cascade classifier, and neither the implementation of those techniques nor the datasets of those studies are available, making unfeasible the reproducibility of their results or the use of their models with new images. On the contrary, using the suite of notebooks presented in the previous section, and a dataset of images provided by the University of Missouri, we have built a freely available YOLO-based stomata detector for images of plant leaves.

The dataset employed to construct our model consists of 468 microscope images of the leaf epidermises of different plant types (including cotton, peanut and maize plants), and those images contain a total amount of 1652 stomata — the dataset is available from the authors on request. The images were annotated by expert biologists using the LabelImg program, that produces annotations in the Pascal VOC format; hence, it was necessary to transform those annotation to the YOLO format using a Python script. After storing the images and labels in the same folder, and fixing the 4 parameters explained in the previous section, the following process was conducted guided by the notebook.

First of all, to improve the generalization of the model, the CLoDSA library was employed to augment the dataset of images by employing techniques like flips, rotations, filters and adding noise. The final dataset was formed by a total of 4212 images, enough for training an object detector. Subsequently, the dataset was split into two sets, using 90% of the dataset for training, and 10% for testing — those sets were automatically split, organized in the corresponding folders, and, additionally, all the configuration files were automatically generated. After training the model for 250000 epochs, we stopped the process and evaluated the model using the testing set. The results achieved by the model were a mAP of 90.91%, a precision of 98% and a F1-score of 99%.

This process can be easily reproduced using the notebook available in the project webpage, and the obtained model can be invoked to detect stomata in images that do not belong neither to the training nor the testing set, see Figure 2. The

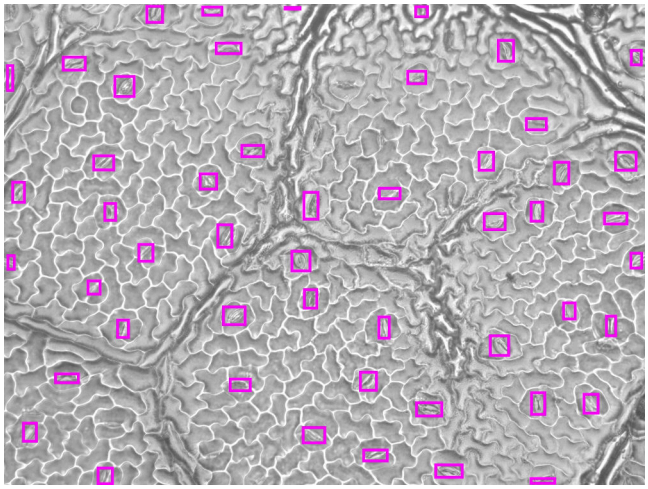


Fig. 2. Stomata identification results

model was trained using a Titan Xp GPU, but it can be employed for detecting stomata in any computer that has Python and C++ installed on it.

V. Conclusions and further work

Democratizing Artificial Intelligence is a movement that has been born with the goal of making artificial intelligence accessible to non-expert users from different fields [36], [37]. The work presented in this paper can be framed in that context; in particular, after carefully analyzing all the steps that are required to construct an object detector using deep learning techniques, we have presented a suite of Jupyter notebooks that allows non-expert users to easily create fast and accurate object-detection models using the YOLO approach — one of the most effective methods for object detection. The suitability of our approach has been tested by developing a model for stomata detection in plant images that achieves a mAP of 90.91%.

In addition to the benefit of simplifying the creation of object detection models, the suite of Jupyter notebooks has value as a teaching material since we have included explanations, both textual and graphical, of all the steps involved in the creation of an object detector; and, hence, they can be useful for Artificial Intelligence and Computer Vision courses.

The main drawback of our suite of plugins is that users need a GPU installed, and properly configured, in their computer, since training deep learning models usually requires the use of a GPU. Therefore, as further work, we plan to integrate our tool in a cloud service like Amazon or Google Cloud to provide a cheap and fast way of constructing object detection models. Moreover, we intend to extend the suite of plugins with other deep learning techniques for object detection, such as SSD or faster R-CNN; and also for other computer vision problems like image classification or semantic segmentation.

References

[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'01)*, vol. 1, 2001, pp. 511–518.

[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, pp. 886–893.

[3] A. Rosebrock, *Deep Learning for Computer Vision with Python*. PyImageSearch, 2017.

[4] W. Liu *et al.*, "Ssd: Single shot multibox detector," in *Proceedings of the 14th European Conference on Computer Vision (ECCV 2016)*, ser. Lecture Notes in Computer Science, vol. 9905, 2016, pp. 21–37.

[5] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the 2014 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'14)*, vol. 1, 2014, pp. 580–587.

[7] R. Girshick, "Fast R-CNN," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV 2015)*, 2015, pp. 1080–1088.

[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Advances in Neural Information Processing Systems*, 2015, vol. 28, pp. 91–99.

[9] S. Ramachandran, J. George, S. Skaria, and V. V. Varun, "Using yolo based deep learning network for real time detection and localization of lung nodules from low dose ct scans," vol. 10575, 2018, p. 105751I.

[10] A. V. Etten, "You only look twice: Rapid multi-scale object detection in satellite imagery," *CoRR*, vol. abs/1805.09512, 2018. [Online]. Available: http://arxiv.org/abs/1805.09512

[11] J. Redmon, "Darknet: Open source neural networks in c," 2013. [Online]. Available: http://pjreddie.com/darknet/

[12] E. Valle *et al.*, "Data, Depth, and Design: Learning Reliable Models for Melanoma Screening," *CoRR*, vol. abs/1711.00441, 2017. [Online]. Available: http://arxiv.org/abs/1711.00441

[13] A. Asperti and C. Mastronardo, "The Effectiveness of Data Augmentation for Detection of Gastrointestinal Diseases from Endoscopical Images," *CoRR*, vol. abs/1712.03689, 2017. [Online]. Available: http://arxiv.org/abs/1712.03689

[14] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[15] M. Everingham *et al.*, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html, 2012.

[16] X. Wang *et al.*, "ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases," in *Proceedings of the 2017 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'17)*, ser. CVPR '17. IEEE Computer Society, 2017.

[17] D. Tzutalin, "LabelImg," 2015. [Online]. Available: https://github.com/tzutalin/labelImg

[18] A. B. Alexey, "YOLO mark," 2018. [Online]. Available: https://github.com/AlexeyAB/Yolo_mark

[19] E. Weill, "Convert datasets," 2017. [Online]. Available: https://github.com/eweill/convert-datasets

[20] P. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Proceedings of the 12th International Conference on Document Analysis and Recognition (ICDAR'03)*, I. C. Society, Ed., vol. 2, 2003, pp. 958–964.

[21] M. D. Bloice, C. Stocker, and A. Holzinger, "Augmentor: An Image Augmentation Library for Machine Learning," *CoRR*, vol. abs/1708.04680, 2017. [Online]. Available: http://arxiv.org/abs/1708.04680

[22] A. Jung, "Imgaug: a library for image augmentation in machine learning experiments," 2017. [Online]. Available: https://github.com/aleju/imgaug

[23] J. Heras *et al.*, "CLoDSA: an open-source image augmentation library for object classification, localization, detection and semantic segmentation," 2018. [Online]. Available: https://github.com/joheras/CLoDSA

[24] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information*

Processing Systems 27, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3320–3328.

[25] W. S. Sarle, "Stopped training and other remedies for overfitting," in *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, 1995, pp. 352–360.

[26] T. Kluyver *et al.*, "Jupyter notebooks — a publishing format for reproducible computational workflows," in *Proceedings of the 20th International Conference on Electronic Publishing*. IOS Press, 2016, pp. 87–90.

[27] R. J. Brunner and E. J. Kim, "Teaching data science," *Procedia Computer Science*, vol. 80, pp. 1947 – 1956, 2016.

[28] K. J. O'hara, D. S. Blank, and J. Marshall, "Computational notebooks for ai education," in *Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2015)*, 2015, pp. 263–268.

[29] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[30] A. Kaehler and G. Bradski, *Learning OpenCV 3*. O'Reilly Media, 2015.

[31] B. R. Buttery *et al.*, "Stomatal numbers of soybean and response to water stress," *Plant and Soil*, vol. 149, no. 2, pp. 283–288, 1993.

[32] P. Giorio, G. Sorrentino, and R. D'Andria, "Stomatal behaviour, leaf water status and photosynthetic response in field-grown olive trees under water deficit," *Environmental and Experimental Botany*, vol. 42, no. 2, pp. 95–104, 1999.

[33] S. Vialet-Chabrand and O. Brendel, "Automatic measurement of stomatal density from microphotographs," *Trees - Structure and Function*, vol. 28, no. 6, pp. 1859–1865, 2014.

[34] S. Liu *et al.*, "A fast method to measure stomatal aperture by mser on smart mobile phone," in *Proceedings of the Imaging and applied optics congress*, 2016, pp. 3–5.

[35] H. Jayakody *et al.*, "Microscope image based fully automated stomata detection and pore measurement method for grapevines," *Plant Methods*, vol. 13, no. 94, 2017.

[36] J. Gao *et al.*, "PANDA: Facilitating Usable AI Development," *CoRR*, vol. abs/1804.09997, 2018. [Online]. Available: http://arxiv.org/abs/1804.09997

[37] Google, "Automl vision," 2018. [Online]. Available: https://cloud.google.com/automl/

Análisis del impacto de datos desbalanceados en el rendimiento predictivo de redes neuronales convolucionales

Francisco J. Pulgar, Antonio J. Rivera, Francisco Charte, María J. del Jesus
Instituto Andaluz Interuniversitario en Data Science and Computational Intelligence (DaSCI)
Departamento de Informática
Universidad de Jaén
Jaén, España
{fpulgar, fcharte, arivera, mjjesus}@ujaen.es

***Abstract*—En los últimos años han surgido nuevas propuestas basadas en Deep Learning para afrontar la tarea de clasificación. Estas propuestas han obtenido buenos resultados en algunos campos, por ejemplo, en reconocimiento de imágenes. Sin embargo, existen factores que deben ser analizados para valorar su influencia en los resultados obtenidos con estos nuevos modelos. En este trabajo se analiza la clasificación de datos desbalanceados con redes neuronales convolucionales (Convolutional Neural Networks-CNNs). Para hacerlo, se han llevado a cabo una serie de tests donde se reconocen imágenes mediante CNNs. Así mismo, se utilizan conjuntos de datos con diferente grado de desbalanceo. Los resultados demuestran que el desequilibrio afecta negativamente al rendimiento predictivo.**

***Index Terms*—Deep Learning, redes neuronales convolucionales, reconocimiento de imágenes, dataset desbalanceado.**

I. Introducción

La tarea de clasificación es una de las más estudiadas dentro del aprendizaje automático, fundamentalmente debido a su gran aplicación para resolver problemas reales. El objetivo principal de esta tarea es obtener un modelo que permita clasificar correctamente nuevos ejemplos, a partir de una serie de instancias previamente etiquetadas [1].

En los últimos años se ha producido un auge en el uso de técnicas basadas en Deep Learning (DL) para afrontar el problema de clasificación. Esto se debe fundamentalmente a dos razones: la gran cantidad de datos disponible y el incremento en la capacidad de procesamiento. Estas técnicas han mostrado muy buenos resultados en clasificación, especialmente en campos como el reconocimiento de imágenes y audio [2], [3].

Uno de los modelos que han obtenido mejores resultados en el reconocimiento de imágenes son las redes neuronales convolucionales (CNNs) [4]. Debido a la naturaleza de la convolución, estas redes se adaptan a la forma en la que se distribuyen los datos de entrada en el caso de las imágenes.

A pesar de los buenos resultados obtenidos con CNNs, son modelos relativamente recientes y existen factores que deben ser estudiados. Uno de ellos es la necesidad de analizar cómo el desbalanceo de los datos afecta al rendimiento predictivo obtenido mediante CNNs. Muchos conjuntos de datos reales contienen algún grado de desbalanceo, de ahí la importancia de estudiar este factor.

Por ello, este estudio se centra en analizar los efectos del desbalanceo de los datos en la clasificación, usando CNNs, de imágenes reales correspondientes a señales de tráfico. En este sentido, se establece la hipótesis de que el rendimiento predictivo se verá afectado negativamente a medida que aumenta el desbalanceo de los datos. La razón que lleva a plantear esta hipótesis es una cuestión de similitud con otras técnicas tradicionales, si el desbalanceo influye negativamente en la clasificación de las redes neuronales tradicionales, podría influir de forma similar a la realizada mediante CNNs. Así mismo, la naturaleza de las CNNs puede verse influenciada por el desequilibrio, ya que el ajuste de los parámetros puede depender del número de ejemplos de cada una de las clases.

El artículo está estructurado de la siguiente forma: La Sección II explica cómo afecta el desbalanceo de los datos a la tarea de clasificación. En la Sección III se introduce el concepto de DL y de CNNs. La Sección IV se pretende verificar la hipótesis establecida, para ello se lleva a cabo una experimentación donde se clasifican conjuntos de imágenes reales mediante CNNs y con diferente grado de desbalanceo. Finalmente, en la Sección V se indican las conclusiones alcanzadas.

II. El problema de desbalanceo en clasificación

La clasificación es una tarea de predicción que, normalmente, utiliza métodos de aprendizaje supervisados [5]. Su propósito es aprender, basándose en datos previos etiquetados, patrones que permitan predecir la clase a asignar a futuros ejemplos que no estén etiquetados. En la clasificación tradicional, los conjuntos de datos están compuestos por una serie de atributos de entrada y un único valor de salida, la clase o etiqueta.

En muchas situaciones reales donde se aplica la clasificación, existen diferencias significativas en el número de elementos correspondientes a las diferentes clases, por tanto, la probabilidad de que un ejemplo pertenezca a cada una de las clases es distinta. Esta situación se conoce como el problema

de desbalanceo [6]–[8]. En muchos casos, la clase minoritaria es la más interesante a la hora de clasificar y tiene un gran coste en caso de no hacerlo correctamente.

Muchos algoritmos de clasificación obtienen buenos resultados cuando trabajan con elementos de la clase mayoritaria, pero los resultados con instancias de la clase minoritaria son erróneos frecuentemente. Esto implica que estos algoritmos, que obtienen buenos resultados con conjuntos de datos balanceados, no obtengan buen rendimiento con datos desbalanceados. Existen diferentes razones para ello:

- Muchas medidas de rendimiento usadas para guiar el proceso de entrenamiento penalizan a las clases minoritarias.
- Las reglas que predicen las clases minoritarias están muy especializadas y su cobertura es muy baja, por ello, a menudo son descartadas en favor de reglas más generales, es decir, aquellas que predicen a las clases mayoritarias.
- El tratamiento del ruido puede afectar a la clasificación de las clases minoritarias, ya que estas clases pueden ser identificadas como ruido y descartadas erróneamente o el ruido existente puede afectar en gran medida la clasificación de las clases minoritarias.

El principal obstáculo es que los algoritmos de clasificación se entrenan con un mayor número de instancias de la clase mayoritaria y cometen más errores cuando intenta clasificar ejemplos de la clase minoritaria. Para afrontar el problema han surgido muchas propuestas que pueden agruparse en [9]:

- **Muestreo de datos:** esta propuesta se basa en modificar el conjunto de entrenamiento proporcionado al algoritmo de clasificación. El objetivo es obtener datos de entrenamiento cuyas clases tengan una distribución más balanceada. En este caso, el algoritmo de clasificación no sufre ninguna modificación [10].
- **Adaptación de algoritmos:** el objetivo perseguido por este tipo de solución es adaptar los algoritmos tradicionales de clasificación para trabajar con datos desbalanceados [11]. En estos casos los datos no son modificados, es el algoritmo el que debe adaptarse.
- **Aprendizaje sensible al coste:** este tipo de solución puede incorporar modificaciones a nivel de datos y de algoritmo. Se basa en incluir penalizaciones más fuertes a los errores cometidos con la clase minoritaria que a los que se producen al clasificar la clase mayoritaria [12].

Al abordar problemas con datos desbalanceados, deben tenerse en cuenta otros factores que pueden tener gran influencia en los resultados obtenidos, por ejemplo, el solapamiento entre clases [13] o el ruido existente en los datos [14].

III. Deep Learning

La necesidad de extraer información de más alto nivel de los datos analizados a través de métodos de aprendizaje ha hecho que emerjan nuevas áreas de estudio, en concreto DL [15]. Los modelos de DL están basados en una arquitectura profunda (multi-capa) cuyo objetivo es mapear las relaciones entre las características de los datos y los resultados esperados [16]. Este tipo de métodos aportan las siguientes ventajas:

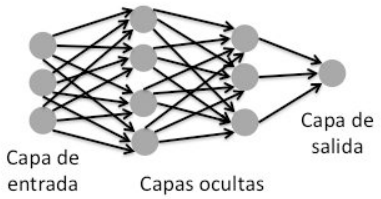
- Los modelos DL incorporan mecanismos para generar nuevas características por sí mismos, sin necesidad de realizar esto en fases externas.
- Las técnicas DL mejoran el rendimiento en cuanto a tiempo de cómputo, al realizar algunas de las tareas más costosas, como la generación de nuevas características.
- Los modelos basados en DL obtienen buenos resultados al afrontar problemas en ciertos campos como reconocimiento de imágenes o sonido, mejorando a técnicas tradicionales [2], [3].

Debido a los buenos resultados obtenidos usando propuestas basadas en DL, se han ido desarrollando diferentes arquitecturas, por ejemplo, CNNs [17] o redes neuronales recurrentes [18]. Estas arquitecturas han sido diseñadas para múltiples campos de aplicación, mostrando una gran eficiencia en el reconocimiento de imágenes. En la Sección III-A, se profundiza sobre el concepto de CNN, modelo que será usado en la experimentación asociada a este estudio.

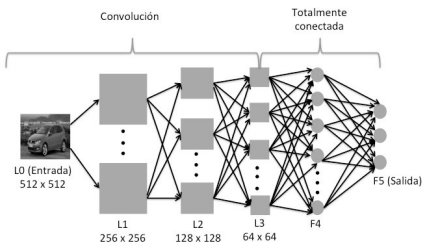
A. Redes Neuronales Convolucionales

Las CNNs son un tipo de red neuronal profunda basada en la forma en la que los animales visualizan e identifican los objetos. Estas redes han mostrado un funcionamiento muy eficiente en ciertos campos de aplicación como en clasificación y reconocimiento de imágenes.

Las CNNs se basan en la idea de la correlación espacial mediante la aplicación de una serie de patrones de conectividad local entre neuronas de capas adyacentes [4], [19]–[21]. Esto implica que, al contrario que otras redes neuronales tradicionales donde cada neurona se conecta con todas las neuronas de las capas anteriores, en las CNNs cada neurona se conecta únicamente a una región concreta de la capa anterior. Otra diferencia fundamental es que las neuronas de las CNNs están distribuidas en tres dimensiones, mientras que las redes tradicionales sólo ocupan dos dimensiones. La Figura 1 muestra la diferencia entre ambos tipos de redes.



(a) Red Neuronal Tradicional



(b) Red Neuronal Convolucional

Fig. 1. Diferencias entre redes neuronales tradicionales y CNNs.

La arquitectura de una CNN está basada en una secuencia de capas, cada una de ellas transforma el volumen de entrada mediante la utilización de una función concreta. Hay tres tipos: capa de convolución, de *pooling* y totalmente conectada. Los tipos de capas indicados anteriormente se usan para formar una CNN compleja, para ello, capas de diferentes tipos se enlazan para formar la arquitectura del modelo que se quiera construir [4], [17], [20], [21].

IV. Experimentación

Una vez expuestos los principales conceptos teóricos necesarios para establecer las bases de este trabajo, se presenta la hipótesis que se pretende verificar y la experimentación que se ha desarrollado para hacerlo.

Durante esta fase se pretende demostrar si un excesivo desequilibrio entre las instancias de las diferentes clases que forman el conjunto de datos afecta al rendimiento predictivo obtenido utilizando CNNs.

Existen diferentes estudios [9]–[12] que muestran que esta propiedad de los datos afecta a determinados modelos de clasificación y proponen soluciones para reducir los efectos al trabajar con datos que presenten desequilibrio. Sin embargo, debido a que las técnicas basadas en CNNs son relativamente recientes, apenas existen estudios que analicen la influencia de los datos desbalanceados en este tipo de modelos.

En este trabajo, se asume la idea de que los datos con desequilibrio entre las clases afectan a la clasificación mediante CNNs. Este hecho lleva a proponer la hipótesis inicial del trabajo: los resultados obtenidos mediante CNNs utilizando datos desbalanceados reducen el rendimiento predictivo de las mismas.

Por tanto, el objetivo de este estudio es analizar si los datos desbalanceados influyen negativamente en la clasificación de imágenes utilizando CNNs. Para hacerlo, se realizan una serie de test usando datos con diferente nivel de desbalanceo (ratio de desbalanceo-IR). Estos datos corresponden a imágenes de tráfico reales [22], siendo el objetivo asociar cada imagen de entrada a la señal correspondiente. El modelo utilizado para realizar la clasificación será una CNN, cuya red usada tendrá la misma arquitectura en todos los experimentos llevados a cabo. Así mismo, debe tenerse en cuenta que no se va a utilizar ningún mecanismo para reducir los efectos del desbalanceo, ya que el objetivo es analizar cómo afectan a estos modelos.

A. Framework experimental

Para desarrollar el experimento se ha utilizado un dataset de señales de tráfico [22] con un total de 11 910 imágenes pertenecientes a 43 tipos de señales o clases diferentes. En primer lugar, es necesario realizar un pre-procesamiento de las imágenes. Esta fase tiene dos objetivos fundamentales: por un lado, recortar la imagen para seleccionar solo la parte correspondiente a la señal de tráfico (Figura 2); y, por otro lado, escalar las imágenes para hacer que todas ellas tengan la misma dimensión. En este sentido, se ha decidido escalar las imágenes a un tamaño de 32x32, ya que es el usado en otros estudios similares [19].

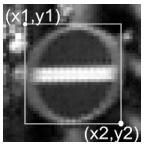


Fig. 2. Ejemplo de imagen recortada (fuente: http://benchmark.ini.rub.de).

Una vez que todas la imágenes del conjunto de datos han sido pre-procesadas, se seleccionan las correspondientes a 10 clases con el objetivo de acentuar el desbalanceo entre los datos. Las clases seleccionadas serán las 5 con más ejemplos y las 5 con menos ejemplos. De esta forma, se obtiene un subconjunto de datos a partir del conjunto completo original.

Una aspecto importante cuando se trabaja con datos desbalanceados es el IR. Esta medida es definida como el ratio entre el número de ejemplos de la clase mayoritaria y el de la clase minoritaria [23], [24].

Otro aspecto a tener en cuenta es que el número de imágenes usadas en cada experimento será el mismo, únicamente cambiará el IR del conjunto de datos. Esto es importante, ya que si el número de ejemplos varía de forma significativa puede afectar a los resultados obtenidos, ocultando así los efectos del desequilibrio de los datos. Por tanto, se han seleccionado 2 700 imágenes para cada ejecución. La razón por la que se selecciona este número viene dada por la cantidad de imágenes de las clases minoritarias en el dataset original. Este conjunto contiene unas 270 imágenes de las clases minoritarias y unas 2 700 de las mayoritarias. Al balancear el conjunto de datos, se seleccionan 270 imágenes de cada clase, por lo que el número total de instancias es 2 700, que se mantiene constante durante todas las ejecuciones, a pesar de cambiar el IR.

Finalmente, se deben indicar que las métricas de evaluación usadas para evaluar el rendimiento predictivo de la CNN en cada caso serán: Tasa de Error, *Precision* y *Recall*.

B. Arquitectura de la CNN

La arquitectura de la CNN usada para clasificar debe ser la misma en todos los casos, a pesar de que el conjunto de imágenes variará en cada uno de ellos. Esto es necesario para evaluar los efectos del desequilibrio en los datos. Esta CNN tiene una arquitectura cuya secuencia de capas es la siguiente:

- **Capa convolucional 1:** 32 filtros son aplicados sobre la imagen original. Tienen un tamaño de 5x5. Esto produce 32 mapas de características.
- **Capa pooling 1:** Los 32 mapas anteriores se reducen usando la función máximo con una ventana de pooling de tamaño 2 y un paso 2.
- **Capa convolucional 2:** Esta capa aplica 64 filtros con un tamaño de 5x5 sobre la salida de la capa anterior y genera un nuevo conjunto de mapas de características.
- **Capa pooling 2:** Se reduce la dimensionalidad de los mapas de características anteriores. Se utiliza la función máximo con una ventana de tamaño 2 y un paso 2.
- **Capa completamente conectada:** En esta capa todos los elementos de la fase anterior son combinados y

usados para realizar la clasificación. Esta capa tiene tantos elementos como clases tenga el problema.

Durante el proceso de entrenamiento, se usa la entropía cruzada para evaluar la red y, posteriormente, la propagación hacia atrás para modificar los pesos de la red. La configuración elegida para la red es la usada por defecto en el software utilizado, TensorFlow [1], considerando el tamaño de las imágenes y los diferentes valores de salida que puede tener el problema.

C. Análisis de resultados

La experimentación llevada a cabo consiste en diferentes ejecuciones en las que se clasifican imágenes reales mediante una CNN, cada una de las cuales tiene diferente IR. En concreto, se han realizado cuatro experimentos con IR 1/10, 1/5, 1/3 y 1/1.

Como se ha descrito en la Sección IV-A, el conjunto de datos seleccionado tiene un total de 11 910 imágenes. Sin embargo, se ha justificado el hecho de seleccionar únicamente 2 700 imágenes en cada uno de los experimentos, con el objetivo de que todas las ejecuciones tengan el mismo número de instancias. Por ello, el primer paso consiste en reducir el número de imágenes de cada clase de forma proporcional y aleatoria, para obtener el número establecido sin afectar a la tasa de desbalanceo. Así, los distintos conjuntos de datos presentan la distribución de ejemplos que puede verse en la Tabla I para cada clase.

Tabla I
INSTANCIAS DE ENTRENAMIENTO Y TEST POR EXPERIMENTACIÓN.

	IR 1/10		IR 1/5		IR 1/3		IR 1/1	
Clase	Train	Test	Train	Test	Train	Test	Train	Test
1	36	11	66	20	98	31	203	67
2	351	115	320	106	288	95	203	67
3	392	130	361	118	325	106	203	67
4	365	121	334	111	301	100	203	67
5	376	125	345	115	311	103	203	67
6	36	11	65	21	97	32	203	67
7	39	13	72	24	108	36	203	67
8	36	11	65	21	97	32	203	67
9	360	120	330	110	297	99	203	67
10	39	13	72	24	108	36	203	67
Total	2030	670	2030	670	2030	670	2030	670

La Tabla I muestra el número de ejemplos de cada una de las clases para los conjuntos de datos de cada ejecución. En ella, se puede ver que todas tienen un total de 2 700 imágenes de las que 2 030 serán usadas para entrenar la red y 670 para evaluar el modelo. Sin embargo, se puede apreciar como el ratio entre las instancias de las clases mayoritarias y minoritarias es diferente. A continuación, se exponen los resultados obtenidos en dichos experimentos.

1) Resultados con IR 1/10:

El primer experimento de clasificación de imágenes utilizando CNN llevado a cabo comienza con un conjunto de datos con un gran nivel de desbalanceo entre clases. La Tabla I muestra el número de instancias de cada una de ellas. Así mismo, se puede ver cómo, en este primer experimento, existe un IR de aproximadamente 1/10 entre las clases minoritarias y mayoritarias. Una vez establecido el conjunto de datos, se utiliza una CNN para realizar la clasificación, obteniendo los resultados presentados en las Tablas II y III.

[1]https://www.tensorflow.org/

Tabla II
NÚMERO DE INSTANCIAS TOTAL Y ERRORES EN TEST POR EXPERIMENTACIÓN.

	IR 1/10		IR 1/5		IR 1/3		IR 1/1	
Clase	Test	Error	Test	Error	Test	Error	Test	Error
1	11	4	20	4	31	2	67	1
2	115	1	106	3	95	2	67	0
3	130	1	118	2	106	2	67	3
4	121	2	111	0	100	2	67	0
5	125	0	115	2	103	0	67	0
6	11	4	21	2	32	2	67	0
7	13	2	24	0	36	0	67	0
8	11	4	21	1	32	0	67	0
9	120	1	110	1	99	1	67	1
10	13	3	24	0	36	0	67	3
Total	670	22	670	15	670	11	670	8

La Tabla II muestra el número de ejemplos de test por clase y el número de errores del modelo al clasificar dichos ejemplos. Además, en la Tabla III se puede ver las métricas Tasa de Error, *Precision* y *Recall* por clase. Ambas tablas presentan los resultados para los 4 experimentos realizados.

En el primero experimento con IR 1/10, los resultados obtenidos muestran que la Tasa de Error por clase tiene un valor global de 0.033, en concreto, de las 670 imágenes de test son clasificadas erróneamente 22. Así mismo, el valor medio de *Precision* es de 0.963 y el de *Recall* 0.848. Estos resultados servirán de base para determinar si las ejecuciones realizadas con menor grado de desbalanceo tienen mejor rendimiento.

2) Resultados con IR 1/5:

El siguiente paso es reducir el IR a 1/5. Para hacerlo, en primer lugar se parte del dataset original con 11 910 imágenes y, posteriormente, se realiza una selección aleatoria del 50% de ejemplos de las clases mayoritarias. De está forma obtenemos un subconjunto con 6 510 imágenes. Una vez hecho esto, se seleccionan 2 700 para que todos los experimentos tengan el mismo número de instancias.

En la Tabla I puede verse el número de ejemplos por clase para este experimento y puede verificarse que el IR es de 1/5. Los resultados obtenidos al clasificar con la CNN esta nueva distribución de ejemplos se muestra en las Tablas II y III.

Estos resultados con un IR 1/5 muestran cómo disminuye la Tasa de Error con respecto a la ejecución anterior. La Tasa de Error obtenida es 0.022, ya que 15 imágenes del total de 670 de test han sido clasificadas erróneamente. Además, los resultados muestran que tanto *Precision* como *Recall* aumentan respecto al experimento previo. El valor medio de *Precision* obtenido es de 0.984 y el de *Recall* es 0.958. Los resultados refuerzan la hipótesis inicial, por lo que se continúa reduciendo el IR.

3) Resultados con IR 1/3:

Para continuar verificando la hipótesis, se reduce el IR a 1/3. Para ello, a partir del dataset de partida con 11 910 imágenes, se selecciona de forma aleatoria el 30% de los ejemplos de las clases mayoritarias, obteniendo un subconjunto con 4 350 imágenes. Posteriormente, se seleccionan 2 700 para que todas las ejecuciones tengan el mismo tamaño.

Tabla III
Resultados para conjunto de test.

	IR 1/10			IR 1/5			IR 1/3			IR 1/1		
Clase	Error	Precision	Recall	Error	Precision	Recall	Error	Precision	Recall	Error	Precision	Recall
1	0.364	**1.000**	0.636	0.200	**1.000**	0.800	0.065	**1.000**	0.935	**0.015**	0.985	**0.985**
2	0.009	0.966	0.991	0.028	0.954	0.972	0.021	0.989	0.979	**0.000**	**1.000**	**1.000**
3	**0.008**	0.963	**0.992**	0.017	0.951	0.983	0.019	0.990	0.981	0.045	**1.000**	0.955
4	0.017	0.983	0.983	**0.000**	**1.000**	**1.000**	0.020	0.990	0.980	**0.000**	0.985	**1.000**
5	**0.000**	0.977	**1.000**	0.017	0.983	0.983	**0.000**	0.956	**1.000**	**0.000**	**0.985**	**1.000**
6	0.364	0.875	0.636	0.095	**1.000**	0.905	0.062	0.968	0.937	**0.000**	0.985	**1.000**
7	0.154	0.917	0.846	**0.000**	0.960	**1.000**	**0.000**	0.947	**1.000**	**0.000**	**1.000**	**1.000**
8	0.364	**1.000**	0.636	0.048	**1.000**	0.952	**0.000**	**1.000**	**1.000**	**0.000**	0.985	**1.000**
9	**0.008**	0.952	**0.992**	0.009	0.991	0.991	0.010	**1.000**	0.990	0.015	0.956	0.985
10	0.231	**1.000**	0.769	**0.000**	**1.000**	**1.000**	**0.000**	**1.000**	**1.000**	0.045	**1.000**	0.955
Media	0.033	0.963	0.848	0.022	0.984	0.958	0.016	0.984	0.980	**0.012**	**0.988**	**0.988**

En la Tabla I, se verifica que la distribución de ejemplos por clase de la tercera experimentación tiene un IR de 1/3, ya que se ha llevado a cabo una reducción mayor de las instancias de la clase mayoritaria. Los resultados obtenidos con esta nueva distribución pueden verse en las Tablas II y III.

Observando los resultados para este experimento con un IR de 1/3, se puede ver que la tendencia vista en la sección previa continúa, ya que mejora el rendimiento. En este caso, la tasa de Error global obtenida es de 0.016, en concreto, se clasifican mal 11 imágenes del total de 670 del conjunto de test. Así mismo, el valor medio de *Precision* es de 0.984 y el de *Recall* es 0.980. De esta forma, se verifica la tendencia general que confirma que, a medida que decrece el IR, los resultados de clasificación mediante CNNs mejoran. Este hecho vuelve a confirmar la hipótesis inicial y, por ello, se pasa a realizar el último experimento con el dataset completamente balanceado.

4) Resultados con IR 1/1:

El objetivo de este último test es, como se ha descrito anteriormente, verificar la mejora de los resultados obtenidos al clasificar con CNN utilizando un dataset balanceado (IR 1/1). Por tanto, el primer paso es balancear el conjunto de datos inicial de 11 910 imágenes. Para hacerlo, se selecciona la clase con menos ejemplos y se eliminan elementos aleatoriamente del resto de clases hasta que tengan el mismo número de instancias. De esta forma, se obtiene un subconjunto de imágenes para llevar a cabo la experimentación con 2 700, cuya distribución puede verse en la Tabla I.

Las Tablas II y III muestran los resultados obtenidos en clasificación usando una CNN y un conjunto de datos balanceado. La Tasa de Error es de 0.012, solo se han clasificado mal 8 imágenes del total de 670 de test, el valor de *Precision* es 0.988 y el de *Recall* 0.988. Estos resultados vuelven a mejorar los obtenidos en las experimentaciones previas.

Las evidencias mostradas por las distintas experimentaciones confirman la hipótesis inicial: a medida que el conjunto de datos está más balanceado, los resultados de la clasificación realizada con CNNs mejoran.

5) Discusión de Resultados:

En las Subsecciones previas, se ha confirmado la hipótesis inicial de este artículo. A continuación se muestra una representación visual de los resultados obtenidos en las Figuras 3 y 4, así mismo, se realiza una discusión de los mismos.

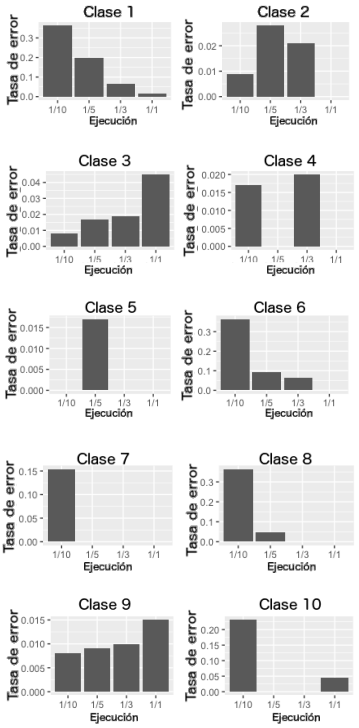


Fig. 3. Tasa de Error por clase y experimento.

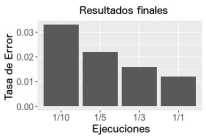


Fig. 4. Tasa media de Error por experimento.

Por un lado, la Figura 3 muestra el Error obtenido por clase para cada experimento, puede verse que en todos los casos, exceptuando las clases 3 y 9, los resultados obtenidos con el conjunto balanceado son los mejores. Por otro lado, la Figura 4 muestra el Error medio para cada experimento, se observa que se obtiene mejor rendimiento a medida que decrece el IR.

Los análisis generales muestran que la reducción del grado de desbalanceo produce una mejora en el rendimiento cuando se clasifica con CNN, lo que confirma la hipótesis inicial. Realizando un estudio por clase, se observa que los mejores resultados se obtienen con el dataset balanceado, excepto en las clases 3 y 9. En estos casos los mejores resultados se obtienen con el conjunto de datos desbalanceado. El motivo puede deberse a que son clases mayoritarias, por lo que al balancear el dataset el número de imágenes de estas clases es menor que en el conjunto de datos desbalanceado, factor que puede afectar al rendimiento predictivo.

Una vez realizado el análisis previo, se concluye que uno de los aspectos que más podría afectar al rendimiento es el cálculo de los pesos de la última capa de la red convolucional. Esta capa completamente conectada determina la clase en una última fase supervisada, y los pesos obtenidos podrían priorizar las clases mayoritarias con respecto a las minoritarias.

Otra característica de la CNN que podría influir es el cálculo de los pesos correspondientes a los diferentes filtros en las capas convolucionales. Estos se mueven a lo largo del espacio de entrada, modificando los pesos durante el proceso, de modo que se podrían adaptar excesivamente a las clases mayoritarias cuando hay un mayor desequilibrio. Estas conclusiones abren nuevas vías de estudio, ya que son necesarios análisis más detallados para verificar si se cumplen o no.

V. Conclusiones

Uno de los principales problemas cuando se afronta la tarea de clasificación con datos reales es que, en muchos casos, no están balanceados, lo que influye negativamente en el rendimiento predictivo de gran cantidad de modelos. En este trabajo, se verifica si este problema de desequilibrio afecta también a la clasificación realizada con CNNs.

Las ejecuciones realizadas han confirmado la hipótesis inicial: a medida que el desbalanceo en los datos es minimizado, los resultados obtenidos a través de la CNN mejoran. Esto implica que debe tenerse en cuenta la distribución de los datos cuando se usan este tipo de técnicas, ya que un excesivo desequilibrio puede afectar negativamente.

Los resultados derivados de este estudio abren nuevas vías de trabajo. Una primera aproximación para afrontar el problema asociado a clasificar datos desbalanceados con CNNs es la aplicación de métodos clásicos: técnicas de muestreo, métodos de aprendizaje sensibles al coste o ensembles, estas técnicas tienen como objetivo reducir los efectos del desequilibrio de los datos. Así mismo, existe la posibilidad de crear nuevos modelos que combinen técnicas tradicionales que afronten el problema de desbalanceo con CNNs, generando algoritmos híbridos que tengan en cuenta este factor.

Este trabajo es una primera aproximación al problema utilizando un conjunto de datos particular y una técnica concreta, por lo que este estudio debe ser ampliado en trabajos futuros para establecer una conclusiones sólidas.

Agradecimientos

Este trabajo de F. Pulgar ha sido financiado por el Ministerio de España de Educación bajo el Programa Nacional FPU (Ref. FPU16/00324). Este trabajo ha sido parcialmente financiado por el Ministerio de España de Ciencia y Tecnología bajo el proyecto TIN2015-68454-R.

Referencias

[1] R. Duda, P. Hart, D. Stork, Pattern Classification, 2nd edition, John Wiley, 2000.
[2] D. Ciresan, U. Meier, Multi-column Deep Neural Networks for Image Classification, Technical Report No. IDSIA-04-12, 2012.
[3] R. McMillan, How Skype Used AI to Build Its Amazing New Language Translator, Wire, 2014.
[4] Y. LeCun, K. Kavukcuoglu, C. Farabet, Convolutional networks and applications in vision, in Circuits and Systems (ISCAS), in Proceedings of 2010 IEEE International Symposium on, p. 253-256, 2010.
[5] S. Kotsiantis, Supervised machine learning: A review of classification techniques, Proceedings of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering, p. 3-24, 2007.
[6] N.V. Chawla, N. Japkowicz, A. Kotcz, Editorial: special issue on learning from imbalanced data sets, SIGKDD Explorations, 6 (1), p. 1-6, 2004.
[7] H. He, E.A. García, Learning from imbalanced data, IEEE Transactions on Knowledge and Data Engineering, 21 (9), p. 1263-1284, 2009.
[8] Y. Sun, A.K.C. Wong, M.S. Kamel, Classification of imbalanced data: a review, International Journal of Pattern Recognition and Artificial Intelligence, 23 (4), p. 687-719, 2009.
[9] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, F. Herrera, A review on ensembles for class imbalance problem: bagging, boosting and hybrid based approaches, IEEE Transactions on Systems, Man, and Cybernetics, 42 (4), p. 463-484, 2012.
[10] G.E.A.P.A. Batista, R.C. Prati, M.C. Monard, A study of the behaviour of several methods for balancing machine learning training data, SIGKDD Explorations, 6 (1), p. 20-29, 2004.
[11] B. Zadrozny, Learning and making decisions when costs and probabilities are both unknown, Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining, p. 204-213, 2001.
[12] B. Zadrozny, J. Langford, N. Abe, Cost–sensitive learning by cost–proportionate example weighting, in Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM'03), p. 435-442, 2003.
[13] Prati, R. C., Batista, G. E., Monard, M. C., Class imbalances versus class overlapping: an analysis of a learning system behavior, in Mexican international conference on artificial intelligence, p. 312-321, 2004.
[14] Frénay, B., Verleysen, M., Classification in the presence of label noise: a survey, IEEE transactions on neural networks and learning systems, 25(5), p. 845-869, 2014.
[15] Y. Bengio, A. Courville, P. Vincent, Representation Learning: A Review and New Perspectives. Pattern Analysis and Machine Intelligence, IEE Transactions, 3 (8), p. 1798-1828, 2013.
[16] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, 2016.
[17] Y. LeCun, Y. Bengio, Convolutional networks for images, speech, and time-series, M. A. Arbib, The Handbook of Brain Theory and Neural Networks, 1995.
[18] H. Sak, A. Senior, F. Beaufays, Long short-term memory recurrent neural network architectures for large scale acoustic modeling, Proc. Interspeech, p. 338-342, 2013.
[19] P. Sermanet, Y. LeCun, Traffic sign recognition with multi-scale convolutional networks, in Proceedings of International Joint Conference on Neural Networks, 2011.
[20] Krizhevsky, A., Sutskever, I., Hinton, G. E., Imagenet classification with deep convolutional neural networks, in Advances in neural information processing systems, p. 1097-1105, 2012.
[21] Jin, K. H., McCann, M. T., Froustey, E., Unser, M., Deep convolutional neural network for inverse problems in imaging, IEEE Transactions on Image Processing, 26(9), p. 4509-4522, 2017.
[22] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition, Neural Networks, vol. 32, p. 323-332, 2012.
[23] V. García, J.S. Sánchez, R.A. Mollineda, On the effectiveness of preprocessing methods when dealing with different levels of class imbalance, Knowledge Based Systems, 25 (1), p. 13-21, 2012.
[24] A. Orriols-Puig, E. Bernadó-Mansilla, Evolutionary rule-based systems for imbalanced datasets, Soft Computing, 13 (3), p. 213-225, 2009.

Comparación de marcos de trabajo de Aprendizaje Profundo para la detección de objetos

Jesús Benito-Picazo, Karl Thurnhofer-Hemsi, Miguel A. Molina-Cabello, Enrique Domínguez
Departmento de Lenguajes y Ciencias de la Computación
Universidad de Málaga
Bulevar Louis Pasteur, 35. 29010 Málaga. España.
{jpicazo, karlkhader, miguelangel, enriqued}@lcc.uma.es

***Resumen*—Muchas aplicaciones en visión por computador necesitan de sistemas de detección precisos y eficientes. Esta demanda coincide con el auge de la aplicación de técnicas de aprendizaje profundo en casi todos las áreas del aprendizaje máquina y la visión artificial. Este trabajo presenta un estudio que engloba diferentes sistemas de detección basados en aprendizaje profundo proporcionando una comparativa unificada entre distintos marcos de trabajo con el objetivo de realizar una comparación técnica de las medidas de rendimiento de los métodos estudiados.**

***Index Terms*—detección de objetos, aprendizaje profundo, redes neuronales convolucionales**

I. Introducción

La detección de objetos es una de las tareas de visión por computador más investigadas, donde en la actualidad las Redes Neuronales Convolucionales (CNNs) están mostrando un altísimo rendimiento. Las CNNs están construidas a partir de muchas capas de neuronas intrínsecamente conectadas en un modelo inspirado por la organización jerárquica de la corteza cerebral del ser humano. Las neuronas actúan como una unidad básica en el aprendizaje y extracción de características de la entrada. El rendimiento del aprendizaje y la extracción de características de la entrada se ve mejorada con el aumento de la complejidad de las redes la cual es causada principalmente por la profundidad de las capas de neuronas. Las técnicas de aprendizaje profundo o Deep Learning (DL) en general, y particularmente las CNNs, son capaces de aprender automáticamente, a partir de imágenes genéricas de entrada, datos con múltiples niveles de abstracción debido a la arquitectura profunda que facilita al modelo el proceso de captura y generalización del mecanismo de filtrado realizando operaciones de convolución en el dominio de la imagen. En la literatura se pueden encontrar muchas CNNs de alto rendimiento tales como AlexNet [1], VGG [2], GoogleNet [3], ResNet [4], etc. Algunas como [5], han demostrado superar la precisión del ojo humano en ciertas tareas de reconocimiento de objetos.

A pesar de la popularidad de otros métodos, los métodos basados en DL están superando a otras técnicas tradicionales de visión por computador por un amplio margen en términos de precisión y algunas veces incluso en eficiencia [6]. No obstante, el estado cambiante del DL producido por una falta de trabajos unificadores y revisiones del estado del arte, hacen la iniciación en este campo tediosa y difícil de mantener actualizada.

El objetivo del presente artículo es realizar un estudio de los marcos de trabajo basados en Deep Learning más prometedores en detección de objetos proporcionando unas medidas de comparación unificadas.

El resto del artículo se organiza como sigue: En la Sección II, se explican las diferentes características de los sistemas de detección y clasificación estudiados, incluidos los conjuntos de datos utilizados para este estudio. La Sección III presenta todas las medidas de rendimiento que han sido utilizadas para realizar los experimentos con los distintos sistemas y los resultados obtenidos de dichos experimentos. La Sección IV se dedicará a la extracción de conclusiones a partir de los resultados obtenidos y a perfilar los siguientes pasos que se darán para mejorar la presente investigación.

II. Metodología

Como ya se comentó en la sección I, el objetivo de este trabajo es realizar una comparación entre diferentes tipos de marcos de trabajo orientados a realizar la detección y localización de objetos basada en redes neuronales profundas. Por tanto, nuestro estudio consistirá en el entrenamiento y prueba de diferentes sistemas fundamentados en modelos de redes neuronales profundas. Como consecuencia, en este trabajo se ha realizado la comparación de cuatro de los citados sistemas: Tensorflow, con su modelo de Faster-RCNN, Pytorch también con su modelo de Faster-RCNN y otros dos sistemas basados en el marco de trabajo de Deep Learning proporcionado por Darknet: uno basado en el modelo YOLO 2.0 (YOLOv2) y otro desarrollado a partir de una combinación del modelo YOLOv2 y el modelo de red neuronal convolucional VGG-16. Dichos modelos serán desglosados con más detalle en las subsecciones II-A, II-B, y II-C, respectivamente. Todos ellos están disponibles para los sistemas operativos Windows, Mac OS X y Linux. La comparación entre los diferentes modelos será llevada a cabo utilizando el conjunto de datos VOC que también será explicado de forma más detallada en la subsección II-D.

II-A. Tensorflow

Publicado en Febrero de 2011 como una evolución del sistema de aprendizaje máquina *DistBelief* desarrollado por Google

Brain, Tensorflow es un sistema de aprendizaje máquina propietario basado en redes neuronales profundas que es capaz de llevar a cabo operaciones en arrays multidimensionales con soporte para ejecutarse en múltiples GPUs y CPUs utilizando sus extensiones CUDA y SYCL. Oficialmente, Tensorflow presenta APIs para los lenguajes C y Python y, de forma no oficial, presenta soporte para los lenguajes C++, Go y Java. Entre las aplicaciones de este sistema podemos encontrar la localización y clasificación en imágenes y el procesamiento del lenguaje natural o NLP. El sistema que vamos a considerar para la realización de nuestro estudio es la implementación de la red neuronal convolucional Faster-RCNN [7] presentada por Tensorflow. Dada la circunstancia de que las Faster-RCNNs trabajan aplicando un clasificador basado en redes neuronales convolucionales a múltiples regiones de una imagen, en nuestro estudio utilizaremos una Faster-RCNN basada en el bien conocido modelo de red neuronal convolucional VGG-16 en las tareas de detección y localización de objetos.

II-B. Pytorch

Pytorch es una biblioteca de código abierto orientada al aprendizaje máquina basada en redes neuronales convolucionales que está programada en Python y construida para su integración profunda con este lenguaje de programación. Proporciona tres características principales de alto nivel: cálculo de tensores, aceleración mediante GPUs y redes neuronales convolucionales construidas sobre un sistema de integración automática.

Pytorch proporciona una gran variedad de rutinas para operaciones con tensores para satisfacer las necesidades de computación aprovechando el uso rápido y sencillo del desarrollo basado en el lenguaje Python. También incorpora Redes Neuronales Dinámicas que utilizan una eficiente implementación de una técnica denominada derivación automática en modo inverso, la cual permite a los usuarios cambiar arbitrariamente la forma en que la red se comporta.

Tal y como sucede con el caso del modelo de Tensorflow, hemos seleccionado como nuestro sistema de detección y clasificación la implementación de la Faster-RCNN desarrollada por Pytorch que está basada en el bien conocido modelo de red neuronal convolucional VGG-16.

II-C. Darknet-YOLO

Darknet es un marco de trabajo desarrollado en lenguaje C++ orientado al diseño, entrenamiento y ejecución de redes neuronales profundas destinadas a la detección y clasificación de objetos en imágenes 2D [8]. Las principales ventajas de este sistema son su simplicidad en términos de uso, tamaño reducido, facilidad de compilación y una documentación en línea clara y concisa. Todas ellas hacen de Darknet-YOLO un sistema fácil de utilizar nada más instalarlo. También es de destacar su capacidad para utilizar el marco de trabajo CUDA de NVIDIA, el cual permite al sistema utilizar la capacidad de cómputo de las GPUs para llevar a cabo tanto procesos de entrenamiento como de validación. Incluido en el marco de trabajo de Darknet, YOLOv2 [9] es un sistema de detección de objetos que aplica una sola red neuronal a la imagen completa utilizando una sola evaluación de la red para dividirla en regiones, las cuales se utilizarán para predecir las localizaciones de los bounding boxes, en contraposición con otros sistemas tales como las Faster-RCNN, cuyo modo de operación consiste en la aplicación de un modelo de CNN conocido a miles de regiones en la misma imagen. Esta diferencia hace a YOLO un sistema mucho más rápido y computacionalmente ligero que los que llevan otros marcos de trabajo tales como Tensorflow o Pytorch, al mismo tiempo que conserva tasas de precisión aceptables en sus predicciones.

En cuestión de detección y localización de objetos basados en CNNs, Darknet es un sistema interesante porque utiliza un algoritmo diferente del modelo clásico de Faster-RCNN pero consigue resultados similares con una menor carga computacional, lo cual le confiere una mayor velocidad. Por esta razón, en la realización del presente trabajo se han considerado dos modelos distintos de redes neuronales para ser utilizadas con este sistema:

- Uno es YOLOV2 que era el modelo más avanzado de YOLO desarrollado por sus autores, J. Redmon y A. Farhadi, en el momento en que se llevó a cabo el presente estudio.
- El otro es una adaptación de YOLOv2 basada en el modelo de CNN VGG-16 que se ha realizado en este mismo trabajo con el objetivo de poder realizar una comparación con los otros basados en las CNN VGG-16 y que ya se citaron en las subsecciones II-A y II-B del presente documento.

Con el objetivo de que que este estudio sea suficientemente fiable y ofrezca unos resultados significativos, es muy importante la selección de un conjunto de datos adecuado que comprenda un conjunto de clases el cual sea lo suficientemente genérico como para representar un buen número de objetos cotidianos, y suficientemente específico como para que ofrezca la posibilidad de extraer conclusiones relacionadas con el rendimiento de cada modelo en las tareas de detección de objetos pertenecientes a clases específicas, y su posibilidad de ser utilizado en determinados entornos. Estas razones han llevado a los autores de este trabajo a seleccionar el popular conjunto de datos VOC2007.

II-D. Los conjuntos de datos VOC

Estos conjuntos de datos fueron originados en las competiciones *PASCAL Visual Object Classes* [10]. El principal objetivo de dichas competiciones era el de reconocer objetos en escenarios tomados del mundo real. Para ese propósito, fueron seleccionadas veinte categorías diferentes de objetos para las cuales se generaron sendos conjuntos de entrenamiento y validación. El número de clases se incrementó de 10 a 20 respecto al proyecto anterior. Así, se consideraron las siguientes clases: persona, pájaro, gato, vaca, perro, caballo, oveja, aeroplano, bicicleta, barco, autobús, coche, motocicleta, tren, botella, silla, mesa de comedor, maceta, sofá y tv/monitor.

De entre todos los conjuntos VOC, los más utilizados son aquellos que se generaron para las competiciones de los

años 2007 [11] y 2012 [12]. El primero de ellos consta de 9963 imágenes que contienen 24640 objetos etiquetados y el segundo contiene 11530 imágenes con un total de 27450 objetos etiquetados. Los conjuntos se encuentran divididos, estando el 50 % dedicado al entrenamiento de los sistemas de clasificación y el 50 % restante, a la validación o prueba de dichos sistemas. Por cada uno de los ficheros de imagen, estos conjuntos de datos contienen un fichero que contiene la localización y dimensiones de las regiones rectangulares mínimas así como la clase asociadas a cada uno de los objetos que se encuentran en dicha imagen. Si el objeto no está claro o está visible menos de un 20 % del mismo, entonces dicho objeto será descartado.

III. Resultados experimentales

En la red pueden encontrarse algunas implementaciones similares del modelo de red Faster-RCNN tanto para el marco de trabajo Tensorflow como para Pytorch. Dadas las similitudes de dichas implementaciones, se ha elegido *tf-faster-rcnn* [13] y *faster-rcnn.pytorch* [14] respectivamente. Para el marco de trabajo de Darknet-YOLO solamente está disponible el código original proporcionado por sus creadores.

Como ya se indicó en la Sección II, con el objetivo de realizar una comparación lo más justa posible entre los diferentes modelos y de obtener resultados lo más significativos posibles para nuestro estudio, se han utilizado los conjuntos de datos VOC2007+VOC2012 para el proceso de entrenamiento y el conjunto de datos VOC2007 para realizar las pruebas de las diferentes propuestas.

Todos los experimentos se han llevado a cabo en un ordenador con una CPU modelo Intel(R) Core(TM) i7-5930K de 64 bits con doce núcleos a 3.50GHz, 64GB de RAM y una GPU NVidia GTX Titan X. El sistema operativo base es Linux-Ubuntu 16.04.3 LTS.

Desde un punto de vista cuantitativo, se han seleccionado diferentes medidas bien conocidas para comparar el rendimiento de la detección y la clasificación de las diferentes propuestas estudiadas. Para ser exactos, se ha considerado la exactitud espacial (S) y la F-medida. Estas medidas proporcionan valores en el intervalo $[0, 1]$, donde mayor es mejor, y representan el porcentaje de aciertos del sistema.

También se consideran en este trabajo los verdaderos positivos (True Positives o TP), falsos negativos (False Negatives o FN), falsos positivos (False Positives o FP), la tasa de falsos negativos (False Negative Rate o FNR), Precisión (Precision o PR) y exhaustividad (Recall o RC). Entre todas estas medidas, la exactitud espacial y la F-medida proporcionan una buena evaluación general del rendimiento de un método dado, mientras que FN debe ser considerado contra FP (menor es mejor), y PR contra RC (mayor es mejor).

La definición de cada medida puede ser descrita como sigue:

$$S = \frac{TP}{TP + FN + FP} \qquad \text{F-medida} = 2 * \frac{PR * RC}{PR + RC} \quad (1)$$

$$RC = \frac{TP}{TP + FN} \qquad PR = \frac{TP}{TP + FP}$$
$$FNR = \frac{FN}{TP + FN} \quad (2)$$

La medida típica para obtener la bondad de un método del tipo Faster-RCNN es la métrica conocida como precisión para la detección de objetos [15]–[17], que utiliza la precisión y la exhaustividad para dibujar la curva precisión-exhaustividad y la precisión media es calculada a partir de las puntuaciones de precisión media para cada clase de objeto.

En este trabajo, la detección y la clasificación son distinguidas y otras evaluaciones han sido consideradas. En este caso, hemos considerado la detección como la habilidad del sistema para localizar un objeto, independientemente de la predicción de la clase del objeto. Por otro lado, la clasificación de los objetos es analizada de acuerdo a su tamaño respecto al tamaño de la imagen.

Por tanto, dada una imagen de test de tamaño $height \times width$ píxeles que considera M objetos reales en la máscara de verdad, con centroide c_gt_m y área a_gt_m para un objeto real $m \in \{1, \ldots, M\}$, y dada una propuesta que estima N objetos predichos, con centroide c_bw_n y área a_bw_n para el objeto predicho $n \in \{1, \ldots, N\}$, la detección del objeto real m es correcta si existe un objeto predicho n con:

$$dist(c\_gt_m, c\_bw_n) <= p \cdot \sqrt{height^2 + width^2} \quad (3)$$

donde *dist* es la función de distancia euclídea y p es una constante de proporcionalidad. Es decir, si la distancia entre ambos centroides es menor que p % del tamaño de la diagonal de la imagen de test entonces el objeto m está correctamente detectado. En este estudio se ha considerado $p = 0{,}03$.

Por otro lado, en el proceso de clasificación también se han considerado objetos lejanos y objetos cercanos. Un objeto real m es lejano si:

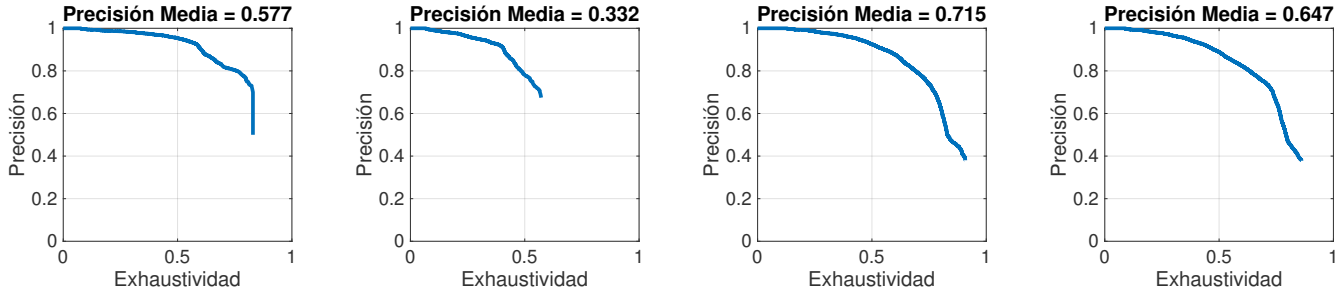
$$a\_gt_m < u \cdot (height \cdot width) \quad (4)$$

donde u es una constante. Un objeto real m es cercano si no es lejano. En este estudio se ha considerado $u = 0{,}05$.

Primero, en la primera fila de la Tabla I se muestra la velocidad de procesado para la tarea de detección de objetos. Estos valores fueron medidos teniendo en cuenta solo la función de detección, que esencialmente hace uso de la GPU a través de CUDA v9.0. Darknet es, como se puede comprobar en la tabla, el mejor marco de trabajo en términos de velocidad de procesado en el proceso de detección. Es seguido por Pytorch, mientras que Tensorflow estaría en la última posición. Es muy interesante remarcar que la diferencia entre los tiempos de computación obtenidos por YOLO y los otros dos marcos de trabajo son significativamente mayores que las diferencias entre Tensorflow y Pytorch. La razón es que Pytorch y Tensorflow están utilizando una implementación basada en una Fast-RCNN, mientras que darknet_yolo utiliza

Cuadro I: Tiempos de computación

	Darknet(yolo)	Darknet(yolo-vgg)	Tensorflow	Pytorch
Media segundos por fotograma	0.009	0.011	0.089	0.048
Media fotogramas por segundo	111.738	90.9	11.297	21.011



(a) darknet_yolo (b) darknet_yolo-vgg (c) tensorflow (d) pytorch

Figura 1: Precisión vs exhaustividad. Las imágenes muestran la figura con la representación del rendimiento en precisión frente a exhaustividad de los diferentes marcos de trabajo (mayor es mejor). La precisión media se calcula como la media de todas las clases de objetos.

un método de evaluación de una sola imagen. Todas las medidas de rendimiento se ven afectadas por este motivo.

Sin embargo, si nos centramos en la exactitud de las detecciones, en la Figura 1 y Tabla II, se puede observar que el mejor rendimiento es obtenido por Tensorflow, seguido por Pytorch. Ambos marcos de trabajo tienen menos falsos negativos en las detecciones de los objetos ya que la exhaustividad es mayor, aunque también es cierto que la precisión es menor que la obtenida por darknet_yolo, lo que indica un mejor comportamiento en términos de falsos positivos para el sistema.

En cuanto al rendimiento de la clasificación, de los resultados obtenidos en los experimentos y mostrados en la Tabla III y Figura 2, se puede observar que, hablando sobre la correcta identificación de los objetos, Tensorflow tiene el mayor número de objetos localizados correctamente clasificados, seguido de Pytorch. También tiene la mayor exactitud media para los objetos localizados correctamente clasificados. Tensorflow también presenta el menor error de distancia medio y la mayor confianza media en la clasificación.

Para algún usuario que quiera utilizar uno de los sistemas presentados en este estudio sería muy útil saber cuántos objetos del total que son detectados, son clasificados erróneamente. A este respecto, comprobando la segunda fila de la Tabla III, se puede ver que el sistema que presenta el menor número de objetos localizados clasificados erróneamente es YOLO-VGG. Parece que el sistema YOLO-VGG es el que comete menos errores en la tarea de identificación una vez que ha detectado un objeto determinado en la imagen.

Las últimas tres filas de la Tabla III muestran una visión general sobre cuántos objetos no han sido identificados por el sistema y la posible correlación de este número con la posición dentro de la imagen y el tamaño de esos objetos respecto al tamaño de la imagen de test. En la tercera fila se muestra el número total de objetos no identificados por el sistema y de los valores que aparecen allí, se puede observar que Tensorflow nuevamente supera a sus competidores mostrando el menor número de objetos no identificados, una vez más seguido de Pytorch. En esta fila también aparece el nombre de la clase de aquellos objetos que permanecen como no identificados más frecuentemente. En este caso, los cuatro sistemas concluyen que, en general, los objetos de la clase "persona" son los que más frecuentemente aparecen como no identificados.

También es interesante conocer dónde están esos objetos no identificados en la imagen de test y su tamaño. En este sentido, la cuarta fila de la Tabla III muestra el número de objetos lejanos no identificados y la clase de los objetos lejanos que más frecuentemente no se identifican. Nuestros experimentos señalan que el sistema con el menor número de objetos no identificados es una vez más Tensorflow con su asociado modelo de red basado en VGG-16. Del mismo modo, los objetos lejanos de clase "botella" son los más frecuentemente no identificados. En la última fila de la Tabla III se muestran los resultados para los objetos cercanos no identificados. Ahí se puede observar nuevamente que Tensorflow supera a sus competidores teniendo el menor número de objetos lejanos no identificados, seguido de Pytorch. Igualmente, se puede observar que los objetos de la clase "persona" son los más frecuentemente no identificados por cada sistema.

IV. Conclusiones

En este trabajo se ha propuesto un estudio de diferentes sistemas de detección y clasificación de objetos en imágenes digitales. Para ello se han realizado pruebas con cuatro sistemas de detección y clasificación basados en redes neuronales convolucionales: Tensorflow, Pytorch, YOLOv2 y YOLOv2-VGG. Se han llevado a cabo pruebas especializadas con el conjunto de datos estándar VOC calculando medidas de rendimiento adaptadas específicamente para la medición del rendimiento en sistemas de detección donde la posición y el

Cuadro II: Rendimiento de detección

	darknet(yolo)	darknet(yolo-vgg)	tensorflow	pytorch
Número de verdaderos positivos	6216	3404	7992	7198
Número de falsos negativos	5816	8628	4040	4834
Número de falsos positivos	1879	1581	6674	6504
Exhaustividad	0.517	0.283	0.664	0.598
Tasa de falsos negativos	0.483	0.717	0.336	0.402
Precisión	0.768	0.683	0.545	0.525
F-medida	0.618	0.400	0.599	0.559
Exactitud	0.447	0.250	0.427	0.388

Cuadro III: Rendimiento de clasificación

	darknet(yolo)	darknet(yolo-vgg)	tensorflow	pytorch
Objetos correctamente identificados				
Número de objetos localizados correctamente clasificados	5974	3253	7531	6738
Exactitud de objetos localizados correctamente clasificados	0.961	0.956	0.942	0.936
Error de distancia media	7.827	8.815	7.096	7.567
Confianza media	0.827	0.805	0.960	0.969
Objetos localizados incorrectamente clasificados				
Número de objetos localizados incorrectamente clasificados	242	151	461	460
Exactitud de objetos localizados incorrectamente clasificados	0.039	0.044	0.058	0.064
Error de distancia media	9.460	9.353	8.451	8.709
Confianza media	0.773	0.748	0.855	0.880
Objetos no identificados				
Número de objetos no identificados	5816	8628	4040	4834
Área media de los objetos no identificados (píxeles)	37199.04	37199.94	48008.87	47350.94
Clase menos identificada (porcentaje de todos los objetos no identificados)	persona (0.156)	persona (0.145)	persona (0.167)	persona (0.164)
Objetos no identificados (lejanos)				
Número de objetos lejanos no identificados	2060	2889	946	1175
Área media de los objetos lejanos no identificados (píxeles)	3772.41	3931,125	3.962	3.977
Clase de objetos lejanos menos identificada (porcentaje de todos los objetos lejanos no identificados)	botella (0.234)	botella (0.231)	botella (0.290)	botella (0.305)
Objetos no identificados (cercanos)				
Número de objetos cercanos no identificados	3756	5739	3094	3659
Área media de los objetos cercanos no identificados (píxeles)	55532.07	53947.39	61476.31	61279.5
Clase de objetos cercanos menos identificada (porcentaje de todos los objetos cercanos no identificados)	persona (0.182)	persona (0.165)	persona (0.191)	persona (0.188)

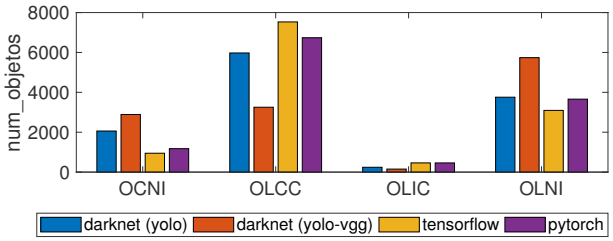


Figura 2: Comparativa de rendimiento de la clasificación de objetos. El número de objetos clasificados por marco de trabajo se muestra de acuerdo a las condiciones consideradas: Objetos Localizados Correctamente Clasificados (OLCC), Objetos Localizados Incorrectamente Clasificados (OLIC), Objetos Cercanos No Identificados (OCNI) y Objetos Lejanos No Identificados (OLNI).

tamaño de un objeto en la imagen de entrada son desconocidos *a priori*.

Los resultados experimentales revelan que Darknet-YOLO es, por una amplia diferencia, el método más rápido de los cuatro siendo un 18 % más rápido que su competidor más directo y el mejor en términos de precisión en la fase de detección. Sin embargo, cuando se trata de precisión en la clasificación, Tensorflow consigue las mejores puntuaciones tanto en la media de la medida de equilibrio precisión-exhaustividad y en número de objetos localizados correctamente clasificados. Los resultados también indican que Pytorch presenta un buen equilibrio entre detección, clasificación y velocidad, lo que lo presenta como un sistema conveniente para su utilización en tareas de detección y clasificación de objetos cuando el usuario está de acuerdo en renunciar a una pequeña cantidad de precisión a cambio de obtener un sistema de detección y clasificación más rápido que funcione casi en tiempo real.

Como una posible extensión del presente trabajo, proponemos extender este estudio a un mayor número de marcos de trabajo para deep learning que actúen como base de sistemas de detección y clasificación de objetos.

Agradecimientos

Este trabajo ha sido financiado parcialmente por el Ministerio de Economía, Industria y Competitividad de España, bajo

los proyectos TIN2014-53465-R y TIN2016-75097-P, y por la Junta de Andalucía, bajo los proyectos TIC-6213 y TIC-657. Todos los proyectos incluyen fondos FEDER. Los autores quieren agradecer los recursos computacionales y asistencia técnica proporcionados por el Servicio de Supercomputación y Bioinformática (SCBI) de la Universidad de Málaga. Así mismo, también quieren agradecer a NVIDIA Corporation por la donación de 2 GPUs Titan X para su investigación. Karl Thurnhofer-Hemsi (FPU15/06512) está disfrutando de una beca de doctorado del Ministerio de Educación, Cultura y Deportes bajo el programa FPU.

Referencias

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the Advances in Neural Information Processing Systems*, 2012, pp. 1097—-1150.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[3] Y. J. P. S. S. R. D. A. D. E. V. V. C. Szegedy, W. Liu and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[4] S. R. K. He, X. Zhang and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2016, pp. 770—-778.

[5] C. J. M. A. G. L. S. G. V. D. D. J. S. I. A. V. P. M. L. e. a. D. Silver, A. Huang, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484—-489, 2016.

[6] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, P. Martinez-Gonzalez, and J. Garcia-Rodriguez, "A Survey on Deep Learning Techniques for Image and Video Semantic Segmentation," *Applied Soft Computing*, vol. 70, pp. 41–65, 2018.

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[8] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

[9] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 6517–6525.

[10] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[11] ——, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results," http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[12] ——, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[13] X. Chen and A. Gupta, "An implementation of faster rcnn with study for region sampling," *arXiv preprint arXiv:1702.02138*, 2017.

[14] J. Yang, J. Lu, D. Batra, and D. Parikh, "A faster pytorch implementation of faster r-cnn," https://github.com/jwyang/faster-rcnn.pytorch, 2017.

[15] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*. Cambridge University Press, 2008, vol. 39.

[16] D. Hoiem, Y. Chodpathumwan, and Q. Dai, "Diagnosing error in object detectors," in *European conference on computer vision*. Springer, 2012, pp. 340–353.

[17] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: An evaluation of the state of the art," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 4, pp. 743–761, 2012.

Adaptación automática del operador de *pooling* aprendiendo pesos de medias ponderadas ordenadas en Redes Neuronales Convolucionales

Juan I. Forcén
Das-nano — Veridas
Poligono Industrial Talluntxe II
Tajonar, España
jiforcen@das-nano.es

Miguel Pagola
dept. Estadística Informática y Matemáticas
Universidad Pública de Navarra
Pamplona, España
miguel.pagola@unavarra.es

Edurne Barrenechea
dept. Estadística Informática y Matemáticas
Universidad Pública de Navarra
Pamplona, España
edurne.barrenechea@unavarra.es

Humberto Bustince
dept. Estadística Informática y Matemáticas
Universidad Pública de Navarra
Pamplona, España
bustince@unavarra.es

***Resumen*—Las Redes Neuronales Convolucionales (RNCs) han logrado ser los modelos de mayor precisión en varias tareas de visión por computador. Tienen estructuras cada vez más complejas en las que se deben seleccionar un buen número de parámetros, como el número de capas, los filtros por capa o las capas de re-muestreo conocidas como *pooling*. En la capa de *pooling* las activaciones de los píxeles vecinos se agregan de forma local; el máximo y la media ponderada son las funciones de agregación comúnmente utilizadas. En este trabajo presentamos una nueva forma de agregar a través de medias ordenadas ponderadas, de tal forma que los pesos de las medias se aprenden durante el entrenamiento de la red. De esta forma el operador de *pooling* se selecciona automáticamente durante el entrenamiento de la red. Por lo tanto, la RNC tiene unos hiper-parámetros menos que no debemos seleccionar. En los resultados experimentales mostramos que la precisión y la capacidad de generalizar de las redes que utilizan este tipo de operador es similar a las redes con operadores de *pooling* fijados.**

***Index Terms*—Clasificación, Redes Neuronales Convolucionales, Pooling, Máximo, Medias ponderadas ordenadas**

I. Introducción

Las Redes Neuronales Convolucionales (RNCs) que están inspiradas en cómo funciona la corteza visual, consisten en múltiples capas de filtros convolucionales de una o más dimensiones [1]. En el ámbito de la clasificación de imágenes las RNCs son el estado del arte en cuanto a precisión y han cambiado el paradigma del procesamiento de imágenes tradicional, ya que las características se aprenden durante el proceso de entrenamiento de la red. Las RNCs son utilizadas en reconocimiento de objetos, conducción automática o reconocimiento del habla [2].

La estructura de una RNC es una serie de capas convolucionales seguidas de capas de *pooling* y finaliza con una serie de capas de neuronas completamente conectadas. El objetivo de cada capa convolucional es producir representaciones que reflejen características locales de la imagen. Cada capa convolucional consta de múltiples canales en los que las características están representadas según los valores de activación. En las capas de *pooling* se agregan las activaciones de cada región de cada canal de la capa de convolución precedente. Este paso se utiliza para obtener una representación más compacta, más robusta e invariante a las transformaciones de las imágenes. A pesar de tener una influencia muy grande en el funcionamiento de la red, la operación de *pooling* no ha sido muy estudiada y la mayor parte de modelos utilizan el máximo o la media aritmética [3]. Al utilizar operadores fijos, en las capas de *pooling* no se aprende ningún parámetro, sin embargo, el tipo de operador de *pooling* que se utiliza en cada una de las capas pasa a ser un hiper-paramétro más de la red. Por lo tanto, pasa a ser otro de los problemas de las RNCs, el alto número de hiper-paramétros [4] (número de capas de la red, número de filtros por capa, el tamaño de los filtros, ratio de aprendizaje, tamaño del subconjunto de entrenamiento, etc.). Debido al alto número de hiper-parámetros, la selección para un problema dado de la arquitectura de red más adecuada tiene un alto coste computacional.

En este trabajo proponemos utilizar como operador de *pooling* medias ponderadas ordenadas [5], cuyos pesos son aprendidos en la fase de entrenamiento. Al ser una media ponderada aseguramos que el resultado de la agregación está siempre entre el máximo y el mínimo valor de los elementos agregados. Por lo tanto, durante del proceso de entrenamiento los valores de los pesos pueden ajustarse a que el operador resultante sea el máximo, la media aritmética o cualquier otro operador que resulte en una red con mejor precisión en el conjunto de entrenamiento.

En la validación experimental hemos comprobado que (en las arquitecturas que hemos probado) si reemplazamos los operadores de *pooling* clásicos por nuestra propuesta, se

alcanza una precisión igual o superior en varios conjuntos de imágenes (CIFAR-10, CIFAR-100, FMNIST).

El trabajo se divide en las siguientes secciones: en la sección II realizamos un análisis de la operación de *pooling* y repasamos los trabajos relacionados. En la sección III describimos el método propuesto y en la sección IV presentamos los resultados obtenidos en los diferentes experimentos. Acabamos con las conclusiones y posibles trabajos futuros.

II. Análisis de la operación de *pooling*

La operación de *pooling*, a veces conocida como de remuestreo, es un paso crucial en varios sistemas de reconocimiento de imágenes como el modelo *Bag-of-Words*, el método VLAD [6], el Super Vector [7] o las RNCs. Dos factores definen la operación de *pooling*: uno es la región de la imagen cuyas características locales se van a agregar y el otro factor es la operación de agregación que define la forma combinar dichas características locales.

En este trabajo nos vamos a centrar en la forma de agregar los valores de las características. Los operadores más comunes son el máximo, utilizado en las arquitecturas de redes más conocidas (AlexNet [8], VGG [9]) y la media aritmética que se utiliza en otros modelos (p.e. en Network in Network [11] o GoogleNet [10]). Recordemos que en la operación de *pooling* simplemente tomamos una ventana de tamaño $n \times n$ de una imagen de características y obtenemos un único valor. Deslizando la ventana de n en n píxeles a lo largo de toda la imagen obtendremos una nueva imagen de menor dimensión que la original.

Boureau et. al. [3], analizó el método conocido como *Spatial Pyramid* [12] e identificó mucho mejor rendimiento en clasificación en varios conjuntos de imágenes utilizando el máximo frente a la media aritmética. Más aún, en [13] Boureau et. al. desarrollaron un estudio teórico en el que demuestran que utilizar el máximo en la fase de agregación de vectores de características es adecuado cuando las características que definen a una clase tienen poca probabilidad de activación. Este resultado fue validado experimentalmente en [14] y en [15]. En ambos trabajos, el operador máximo obtiene los mejores resultados cuando se utilizan vectores dispersos de características muy grandes. En los vectores de características dispersos, cada característica tiene poca probabilidad de activación (es decir que su valor sea mayor que cero) y tiene mucho poder discriminatorio. Sin embargo, cuando se utilizan operadores de *pooling* que suavizan el valor máximo (el máximo esperado en [13] y otras metodologías en [14]) se obtienen todavía mejores resultados que con el máximo. Similar resultado obtuvimos en [16], donde estudiamos la cardinalidad del operador de *pooling* que realiza la media sobre los $\mathcal{N}$ valores mayores (utilizando *Bag-of-Words* y *Spatial Pyramid*). Además en [17] comprobamos que las medias ponderadas ordenadas también dan buenos resultados en dicha metodología de clasificación de imágenes. En [14] y [13] se argumenta que las conclusiones obtenidas se pueden trasladar a la operación de *pooling* de las RNCs, ya que si aplicamos el método de *Bag-of-Words* en diferentes capas, el modelo global es equivalente a una RNC.

Pensemos en una red con una estructura con varias capas. Si en la imagen que estamos tratando existe, por ejemplo, una esquina muy definida, habrá un valor de activación muy alto en una imagen de características en la que se representen las esquinas. Por lo tanto, si en las restantes capas de *pooling* utilizamos el operador máximo, este valor (que es muy alto) se va a ir propagando por la arquitectura de la red y llegará a formar parte de la representación final de características de la imagen. Si dicha esquina es representativa de la clase de la imagen, entonces será una característica discriminatoria y servirá para clasificar correctamente la imagen.

Por otro lado, imaginemos que por toda la imagen hay una textura que no está claramente definida, pero que representa a la clase de la imagen. El valor de activación en la imagen de características de esa textura será pequeño. Si el operador de *pooling* en todas las capas es el máximo, el valor que se propaga por la red es pequeño y por lo tanto desaparecerá o no tendrá mucha representación en el vector final. Sin embargo, si el operador utilizado es la media aritmética, al aparecer la textura por toda la imagen, su representación en el vector final será mayor. Esto es debido a que las características que aparecen aisladamente son filtradas por la media y acaban teniendo un valor más pequeño en la representación final.

Teniendo en cuenta que el máximo y la media son operadores adecuados para diferentes tipos de características, en [18] se proponen dos métodos en los que se aprende la función de *pooling*. En la primera estrategia, se aprende un parámetro que mezcla el resultado del valor máximo con la media aritmética. En el segundo método se aprende una función de *pooling* en forma de árbol que va mezclando los resultados de diferentes filtros de *pooling*. Estos filtros también se aprenden y son idénticos a los filtros de convolución. En ambos casos obtiene mejores resultados que si se utilizan operadores de *pooling* fijos. Sin embargo, la segunda estrategia es similar a añadir nuevas capas de convolución, por lo tanto, no puede considerarse una generalización de la operación de *pooling*. En este trabajo proponemos un método de pooling en el que se aprenda una función de *pooling*. El objetivo es aprender unos coeficientes que afecten únicamente a los valores de las activaciones (en el caso de la convolución los pesos están asociados a su posición espacial en el filtro). Para ello utilizaremos medias ordenadas ponderadas, en las que el vector de valores se ordena de mayor a menor y después cada valor es multiplicado por el coeficiente asociado a cada posición del vector ordenado. Los valores de los pesos se aprenderán en la fase de entrenamiento. De esta forma, nos proponemos obtener un operador de *pooling* que propague los valores más altos de las activaciones, teniendo en cuenta también los valores de activaciones medios que aparezcan frecuentemente.

III. *Pooling* basado en medias ponderadas ordenadas

Los operadores estándar de *pooling* son o el máximo $f_{max}(\boldsymbol{x}) = \max_{i \in N}\{\boldsymbol{x}_i\}$ o la media aritmética $f_{med} =$

$\frac{1}{N}\sum_{i\in N}\boldsymbol{x}_i$, donde el vector $\boldsymbol{x}$ contiene los valores de activación de una región de *pooling* local de tamaño N píxeles (típicamente 2x2 o 3x3).

Las medias ponderadas ordenadas son funciones de agregación promedio. Se diferencian de las medias ponderadas en que los pesos no están asociados a unas posiciones del vector de entrada sino a las magnitudes. Fueron propuestas por Yager en 1988 [5].

$$f_{mpo}(\boldsymbol{x}_i \searrow) = \sum_{i\in N} w_i x_i \tag{1}$$

Donde $(\boldsymbol{x}_i \searrow)$ es un vector ordenado con $x_1 \geq x_2 \geq \cdots \geq x_N$. La suma de pesos debe ser igual a uno $\sum_{i=1}^{n} w_i = 1$ y $w_i \geq 0$ para cada $i = 1, ..., n$.

El cálculo de la media ponderada ordenada requiere de la ordenación de los valores que van a ser agregados. Dependiendo de los valores de los pesos podemos obtener las funciones de agregación típicas [19], por ejemplo:

- Si $w = (0, 0, \cdots, 0, 1)$, entones f_w = mínimo.
- Si $w = (1, 0, \cdots, 0, 0)$, entones f_w = máximo.
- Si $w = (\frac{1}{n}, \frac{1}{n}, \cdots, \frac{1}{n}, \frac{1}{n})$, entonces f_w = media aritmética.
- Si $w = (0{,}5, 0{,}5, 0, 0, 0, 0, 0, 0, 0, 0, 0)$, entonces f_w = media de los dos valores mayores.

Cuando utilizamos un operador de *pooling* con medias ponderadas ordenadas tenemos varias posibilidades, aprender unos pesos por red, por capa, por cada canal de cada capa o incluso de cada región de cada canal de cada capa. A continuación vamos a desarrollar las ecuaciones para el caso de aprender un conjunto de pesos en una capa de la red. Sea la función de coste de la red J, creamos una nueva función de coste en la que añadimos tres términos. El primer término fuerza que los valores sean mayores que cero, el segundo obliga a que la suma de pesos sea 1 y el tercer término penaliza las diferencias entre los pesos consecutivos. Al utilizar estos términos (similares a una regularización) obtenemos unos valores de los pesos que serán más interpretables.

$$J_{mpo} = J + C_1 \textstyle\sum_{i=1}^{N} max\{0, -w_i\} +$$
$$+ \quad C_2 \left(\left(\textstyle\sum_{i=1}^{N} w_i\right) - 1\right)^2 + C_3 \left(\textstyle\sum_{i=1}^{N-1} (w_i - w_{i+1})^2\right)$$

Por lo tanto el cálculo del gradiente para utilizarlo en el algoritmo de optimización queda:

$$\Delta_{w_i} J_{mpo} = \Delta_{w_i} J - C_1 +$$
$$+2C_2 \left(\left(\textstyle\sum_{i=1}^{N} w_i\right) - 1\right) w_i + 2C_3(w_i - w_{i+1}) \tag{2}$$

Si $w_i \geq 0$ el término C_1 desaparece. El cálculo del gradiente $\Delta_{w_i} J$ es similar al de una capa de convolución. Teniendo en cuenta que en lugar de la convolución, hay que ordenar los valores de la capa de activación.

IV. RESULTADOS EXPERIMENTALES

En la fase de experimentación queremos comprobar la precisión de RNCs ya conocidas y validadas cuando sustituimos el operador de *pooling* original por el operador de *pooling* basado en medias ordenadas ponderadas. Además queremos comprobar si los pesos convergen hacia unos operadores de *pooling* similares a los operadores originales ($(1, 0, 0, \cdots, 0)$ para el máximo o $(\frac{1}{n}, \frac{1}{n}, \cdots, \frac{1}{n})$ para la media). Los modelos de red que utilizaremos son dos: la red propuesta en [11] que es conocida como Network in Network (NiN) y el modelo VGG [9]. Ambas arquitecturas quedan reflejadas en la tabla I y el tabla II respectivamente (todas las convoluciones van seguidas de activaciones tipo Relu).

Cuadro I
PARÁMETROS DE LA RED NIN

Input		Filtros, canales
32x32		5x5, 192
32x32		1x1, 160
32x32		1x1, 96
32x32	pool1	3x3 Max *pooling*, stride 2
16x16		dropout 0.5
16x16		5x5, 192
16x16		1x1, 192
16x16		1x1, 192
32x32	pool2	3x3 Med *pooling*
8x8		dropout 0.5
8x8		5x5, 192
8x8		1x1, 192
8x8		1x1, 10 o 100
8x8	pool3	8x8 Med *pooling*
10 o 100		Softmax

Cuadro II
PARÁMETROS DE LA RED VGG

Input		Filtros, canales
28x28		3x3, 64
28x28		3x3, 64
28x28	pool1	2x2 Max *pooling*, stride 2
14x14		3x3, 128
14x14		3x3, 128
14x14	pool2	2x2 Max *pooling*, stride 2
7x7		3x3, 256
7x7		3x3, 256
7x7	pool3	2x2 Max *pooling*, stride 2
4x4		3x3, 512
4x4		3x3, 512
4x4	pool4	2x2 Max *pooling*, stride 2
2x2		3x3, 512
2x2		3x3, 512
2x2	pool5	2x2 Max *pooling*, stride 2
-		Flatten
-		2048
-		512
-		10
-		Softmax

Hemos utilizado los conjuntos de imágenes CIFAR-10, CIFAR-100 [20] y Fashion-MNIST [21]. Los datasets CIFAR-10 y CIFAR-100 contienen 50000 imágenes a color para la fase de entrenamiento y 10000 de test. Tienen 10 y 100 categorías respectivamente, mientras que FMNIST contiene 60000 imágenes en escala de grises para entrenamiento y 10000 de test (10 categorías). Las imágenes de CIFAR-10 y

CIFAR-100 tienen una resolución de 32x32 píxeles, mientras que las de FMNIST de 28x28.

En la tabla III mostramos los resultados obtenidos para la red NiN en los conjuntos de datos CIFAR-10 y CIFAR-100. En la primera fila se muestra el porcentaje de acierto en los conjuntos de test. En la fila Max se muestra el resultado cuando los operadores de todas las capas son *pooling* el máximo. En la fila Med todos los operadores de *pooling* son la media aritmética. Las filas con MPO1 corresponden a cuando entrenamos en todas las capas de *pooling* una media ponderada ordenada. En el caso MPO2 entrenamos un operador por cada canal de cada capa. En la fila MPO1ft, se muestran los resultados cuando entrenamos la red con todos los operadores de *pooling* con la media aritmética y cuando el entrenamiento converge, se sustituyen los operadores de *pooling* por medias ponderadas ordenadas, se fijan los pesos de las capas convolucionales y se entrena durante unas pocas épocas para que se ajusten los pesos de las capas de *pooling*. Este método es similar al conocido *fine tunning*. En el resto de filas el sufijo pl significa que el entrenamiento de los pesos del operador de *pooling* se realiza sin añadir los términos de regularización a la función de coste. De tal manera que los pesos convergen a valores que no cumplen las condiciones para ser una media ponderada (es decir, no tienen que ser positivos ni sumar 1). En la tabla IV se muestran los resultados de la red VGG en FMNIST (en el modelo original todos los operadores de *pooling* son el máximo).

Observando los resultados comprobamos que los modelos en los que se utilizan las MPOs alcanzan una precisión parecida al modelo original. Un poco menor en VGG y un poco mayor en el caso de NiN para CIFAR-100. Cuando se entrenan los pesos libres la precisión es un poco mayor a cuando los pesos cumplen las condiciones de las MPOs. También se comprueba que ajustar un operador de *pooling* por cada canal de cada capa no es necesario, de hecho la precisión de MPO2 es más baja que MPO1 en todos los casos. Por lo tanto, podemos utilizar las MPOs para encontrar modelos de red sin tener que identificar manualmente cuáles son los operadores de *pooling* adecuados para cada capa.

Cuadro III
Resultados experimentales de la red NiN

NIN	CIFAR-10	CIFAR-100
Orig	90.24	58.70
Max	88.76	54.89
Med	90.50	58.75
MPO1	90.34	59.33
MPO2	90.32	58.09
MPO1ft	90.39	59.42
MPO1pl	**90.58**	**59.66**
MPO2pl	90.36	58.77
MPO1pl-ft	90.46	59.62

IV-1. Test robustez: Al añadir más parámetros a la red que se entrenan cabe pensar que obtendremos redes que están más ajustadas y por tanto con menos capacidad de generalización. Para comprobar este hecho hemos realizado un test de robustez

Cuadro IV
Resultados experimentales de la red VGG.

VGG	FMNIST
Max	**94.31**
Mean	92.90
MPO1	93.57
MPO2	92.11
MPO1ft	92.87
MPO1pl	94.25
MPO2pl	92.91
MPO1pl-ft	92.770

[18]. En este test, se rotan las imágenes de test entre -8 y 8 grados y se comprueba la precisión (Figuras 1, 2, 3). Comprobamos que en el caso de NiN, el modelo con MPO es el más robusto a los cambios y en VGG es prácticamente igual al modelo original en el que todos los operadores de *pooling* son el máximo. Por lo tanto, ponderar las activaciones teniendo en cuenta únicamente su valor, provoca que las características representativas de la imagen se propaguen por la red hasta la representación final y mejoren la clasificación.

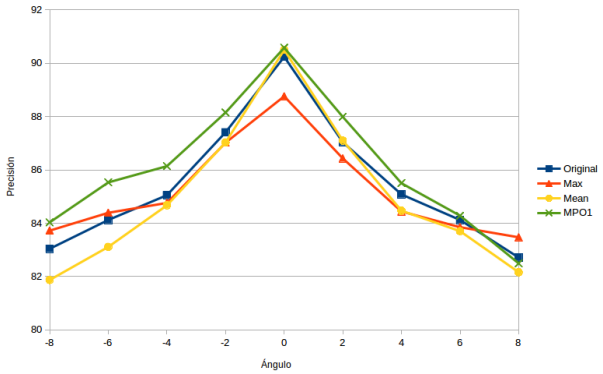


Figura 1. Test de Robustez giro NiN en CIFAR-10.

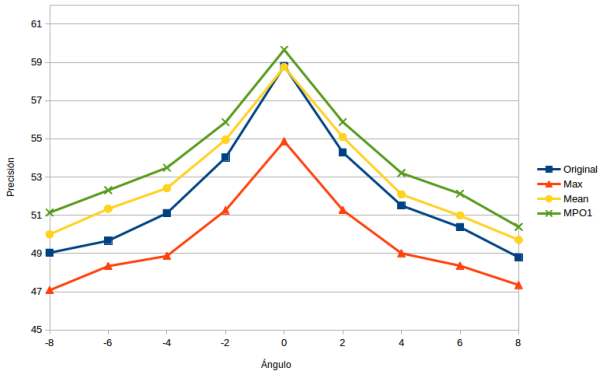


Figura 2. Test de Robustez giro NiN en CIFAR-100.

IV-2. Análisis de los pesos: En la figura 4 vemos los 9 pesos de las dos primeras capas de *pooling* de la red NiN en el caso MPO1 cuando entrenamos la red para CIFAR-10 y CIFAR-100 respectivamente. En el modelo original los

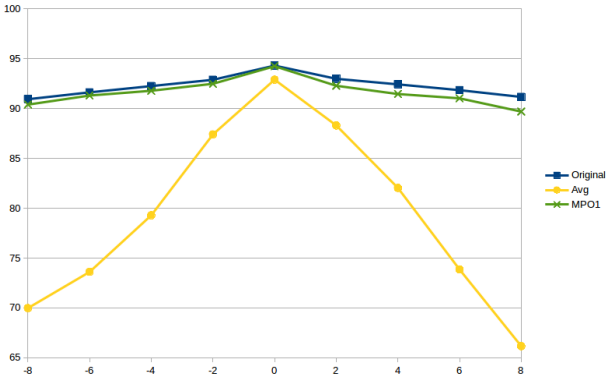


Figura 3. Test Robustez giro VGG en FMNIST.

operadores utilizados son el máximo para la primera capa y la media aritmética para la segunda capa de *pooling*. El coeficiente 1 multiplica al valor mayor y así sucesivamente. En el caso de CIFAR-100, vemos que todos los pesos son prácticamente iguales (ligeramente descendientes), por lo tanto el resultado de estos operadores será muy similar a la media aritmética. Sin embargo, al entrenar la red con CIFAR-10 los coeficientes de cada capa son diferentes. Aunque en ambos casos se obtienen coeficientes mayores para los valores más pequeños, es decir que el valor resultante será incluso más pequeño que la media.

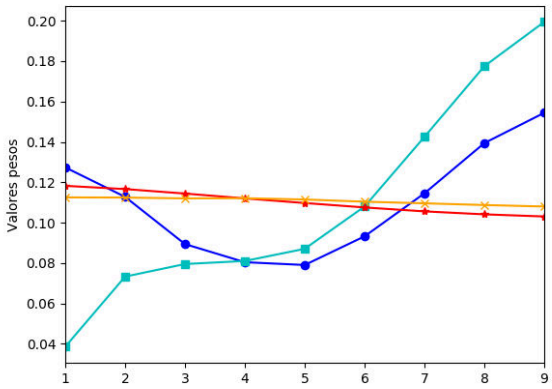


Figura 4. Azul: pesos de la capa pool1 entrenando en CIFAR-10; Cyan pesos de la capa pool2 entrenando en CIFAR-10; Rojo: pesos de la capa pool1 entrenando en CIFAR-100; Naranja pesos de la capa pool2 entrenando en CIFAR-100.

En la figura 5 vemos los 64 pesos de la tercera capa de *pooling* de la red NiN en el caso MPO1 cuando entrenamos la red para CIFAR-10 y CIFAR-100 respectivamente. Los pesos obtenidos con CIFAR-10son similares a las capas pool1 y pool2, los pesos más grandes multiplican a los valores pequeños. Esto significa que se da importancia a las activaciones de menor valor. Para el caso de CIFAR-100 los pesos son parecidos a la media aritmética (los valores negativos se deben a que el factor de regularización no ha penalizado lo suficiente y el entrenamiento ha convergido sin llevar ese término a cero).

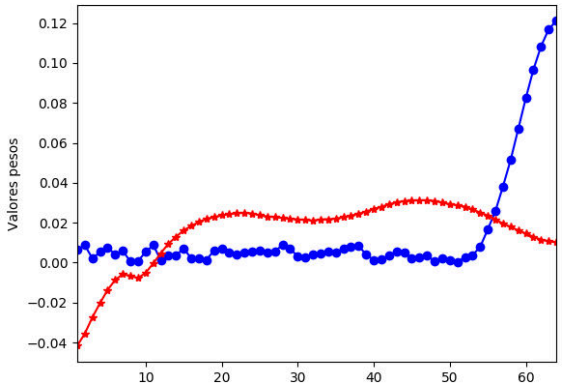


Figura 5. Azul: pesos de la capa pool3 entrenando en CIFAR-10; Rojo: pesos de la capa pool3 entrenando en CIFAR-100.

Al fijarnos en la precisión obtenida por los modelos vemos que MPO1 para CIFAR-100 es bastante mejor que el modelo original y en CIFAR-10 es un poco peor.

En la figura 6 mostramos los 4 pesos obtenidos para las 5 capas de *pooling* de la red VGG (en el modelo original todos los operadores son el máximo). En este caso las capas 1, 3, 4 y 5 convergen a un operador que da más peso a los valores más altos. Sin embargo, los pesos de la segunda capa dan más importancia a los valores pequeños. Puede ser que este sea el motivo de que la precisión del modelo MPO1 es menor que la del modelo original. Por lo tanto, nos queda por comprobar si al lanzar más entrenamientos obtenemos modelos que obtienen una precisión mejor y cuales son los valores de los pesos.

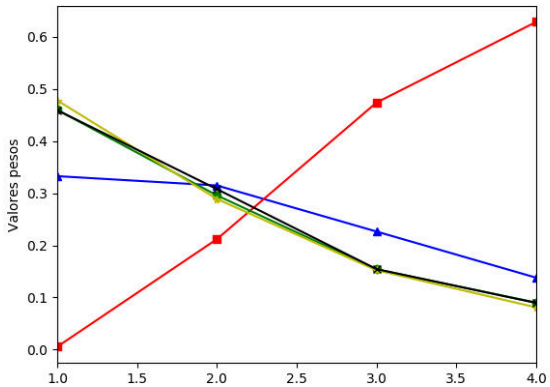


Figura 6. Pesos de las capas pool1 (en azul), pool2 (en rojo), pool3 (en verde), pool4 (en amarillo) y pool5 (en negro) de la red VGG (MPO1) para FMNIST.

IV-3. Análisis de los tiempos: El mayor problema de las medias ponderadas ordenadas es la necesidad de ordenar el vector de valores que queremos agregar. Al introducir esta ordenación en la fase de entrenamiento, hace que los tiempos de

computación crezcan. En el tabla V mostramos la proporción de tiempo de los diferentes modelos con respecto al original. El costo en tiempo del modelo MPO1 es demasiado grande y probablemente sea más rápido probar dos o tres combinaciones de operadores de *pooling* basándonos en nuestra experiencia. Sin embargo, si entrenamos con todos los operadores siendo la media y luego realizamos el ajuste de los pesos de los operadores, simplemente doblamos el tiempo del modelo original y obtenemos un modelo robusto y con una precisión similar al mejor modelo. Por lo tanto, si utilizamos esta metodología, el uso de medias ponderadas puede utilizarse en casos reales en los que no conozcamos la arquitectura correcta e introducimos en la fase de entrenamiento la selección del operador de *pooling*.

Cuadro V
TIEMPOS DE ENTRENAMIENTO

NiN	Orig	Max	Med	MP01	MPO1ft
Tiempo	1	1.02x	0.95x	4.06x	1.69x
VGG	Orig	Max	Med	MPO1	MPO1ft
Tiempo	1	1	0.83x	7.26x	2.11x

V. Conclusiones y trabajos futuros

En este trabajo hemos propuesto aprender los pesos de medias ponderadas ordenadas para adaptar automáticamente el operador de *pooling*. De esta forma eliminamos un hiperparámetro a la hora de seleccionar la estructura de la red. En las pruebas experimentales hemos comprobado que añadiendo muy pocos parámetros, aprendiendo un único operador por capa, el *pooling* basado en medias ponderadas obtiene una precisión similar a los modelos originales, teniendo la red obtenida una mayor precisión frente a variaciones de las imágenes de test. Como trabajo futuro falta por analizar las condiciones iniciales y los parámetros del entrenamiento para comprobar que se ha convergido a la mejor solución o si es posible obtener otros pesos que mejoren la precisión de la red. Otro posible estudio consiste en relacionar el operador de *pooling* a la zona de la imagen original. Teniendo en cuenta que la estructura de la imagen original se mantiene a lo largo de las capas de la red, en lugar de aprender un operador por cada canal, el objetivo es aprender un operador por cada zona.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el proyecto TIN2016-77356-P (AEI/FEDER, UE).

Referencias

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, pp. 541–551, Dec 1989.

[2] N. Aloysius and M. Geetha, "A review on deep convolutional neural networks," in *2017 International Conference on Communication and Signal Processing (ICCSP)*, pp. 0588–0592, April 2017.

[3] Y. L. Boureau, F. Bach, Y. LeCun, and J. Ponce, "Learning mid-level features for recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2559–2566, 2010.

[4] Y. Bengio, *Practical Recommendations for Gradient-Based Training of Deep Architectures*, pp. 437–478. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[5] R. Yager, "On ordered weighted averaging aggregation operators in multi criteria decision making," *IEEE Trans. Syst. Man Cybern.*, vol. 18, no. 1, pp. 183–190, 1988.

[6] F. Perronnin, M. Douze, S. Jorge, C. Schmid, F. Perronnin, M. Douze, S. Jorge, and P. Patrick, "Aggregating local image descriptors into compact codes Aggregating local image descriptors into compact codes," 2012.

[7] X. Zhou, K. Yu, T. Zhang, and T. S. Huang, "Image classification using super-vector coding of local image descriptors," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6315 LNCS, no. PART 5, pp. 141–154, 2010.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, (USA), pp. 1097–1105, Curran Associates Inc., 2012.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, June 2015.

[11] M. Lin, Q. Chen, and S. Yan, "Network in network," *CoRR*, vol. abs/1312.4400, 2013.

[12] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 2169–2178, 2006.

[13] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A Theoretical Analysis of Feature Pooling in Visual Recognition," *Icml*, pp. 111–118, 2010.

[14] P. Koniusz, F. Yan, and K. Mikolajczyk, "Comparison of mid-level feature coding approaches and pooling strategies in visual concept detection," *Computer Vision and Image Understanding*, vol. 117, pp. 479–492, may 2013.

[15] C. Wang and K. Huang, "How to use Bag-of-Words model better for image classification," *Image and Vision Computing*, vol. 38, pp. 65–74, 2015.

[16] M. Pagola, J. I. Forcen, E. Barrenechea, J. Fernández, and H. Bustince, "A study on the cardinality of ordered average pooling in visual recognition," in *Pattern Recognition and Image Analysis* (L. A. Alexandre, J. Salvador Sánchez, and J. M. F. Rodrigues, eds.), (Cham), pp. 437–444, Springer International Publishing, 2017.

[17] M. Pagola, J. I. Forcen, E. Barrenechea, C. Lopez-Molina, and H. Bustince, "Use of owa operators for feature aggregation in image classification," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–6, July 2017.

[18] C. Y. Lee, P. Gallagher, and Z. Tu, "Generalizing pooling functions in cnns: Mixed, gated, and tree," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, pp. 863–875, April 2018.

[19] G. Beliakov, H. B. Sola, and T. C. Sánchez, *A Practical Guide to Averaging Functions*. Springer, 2016.

[20] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., 2009.

[21] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017.

Uso de técnicas de Saliency para Selección de Características

Brais Cancela
LIDIA Group
Universidade da Coruña
A Coruña, Spain
brais.cancela@udc.es

Verónica Bolón-Canedo
LIDIA Group
Universidade da Coruña
A Coruña, Spain
veronica.bolon@udc.es

Amparo Alonso-Betanzos
LIDIA Group
Universidade da Coruña
A Coruña, Spain
ciamparo@udc.es

João Gama
LIAAD Group
University of Porto
Porto, Portugal
jgama@fep.up.pt

***Resumen*—Las técnicas clásicas de selección de características buscan la eliminación de aquellas características que son irrelevantes o reduntantes, obteniendo un subconjunto de características relevantes, que ayudan a una mejor extracción de conocimiento del problema a tratar, permitiendo modelos de aprendizaje más sencillos y fáciles de interpretar. La gran mayoría de estas técnicas trabajan sobre el conjunto completo de datos, pero no nos proporcionan una información detallada caso por caso, que es el escenario del que partimos en esta propuesta. En este trabajo mostraremos un nuevo método de selección de características, basado en las técnicas de saliency en deep learning, que es capaz de proporcionar un conjunto de características relevantes muestra a muestra, y finalmente proporcionar un subconjunto final de las mismas.**

***Index Terms*—selección de características, deep learning, saliency**

I. Introducción

Con el auge del denominado *Big Data*, el uso de técnicas que permitan reducir la dimensionalidad del espacio de entrada se hace cada vez más necesario. Las técnicas que acometen esta tarea se dividen, habitualmente, en dos grandes grupos: la *selección de características* (feature selection) y la *extracción de características* (feature extraction). En la Figura 1 podemos ver una representación gráfica sobre su funcionamiento.

Por una parte, las técnicas de extracción de características reducen el número de características mediante la combinación de las variables originales. Un ejemplo en *Deep Learning* serían las conocidas como *deep features*, que son las características que se utilizan como entrada de la última capa de una red. De esta manera, estas técnicas son capaces de crear un nuevo conjunto de características, que suele ser más compacto y con una capacidad mayor de discriminación. Esta es la técnica más utilizada en análisis de imágenes, procesamiento de señales o en recuperación de la información.

Por otra parte, las técnicas de selección de características consiguen la reducción de la dimensionalidad mediante la eliminación de características tanto irrelevantes como redundantes. Debido al hecho de que la selección de características mantiene los atributos *originales*, es una técnica especialmente útil para aplicaciones donde los atributos originales son importantes para la interpretación del modelo y la extracción de conocimiento.

Brais Cancela agradece el apoyo de la Xunta de Galicia bajo su programa postdoctoral. También agradecemos su ayuda a NVIDIA, que nos ha facilitado una tarjeta Titan Xp bajo el 'GPU Grant Program'.

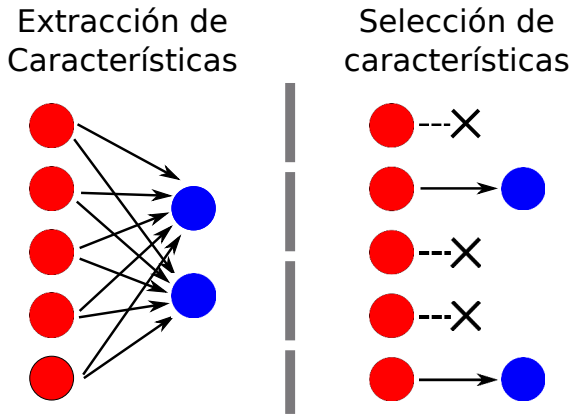


Figura 1. La extracción de características crea nuevas características mediante la combinación del conjunto inicial, mientras que la selección de características elimina aquellas que considera irrelevantes o redundantes, manteniendo inalteradas las restantes.

Uno de los problemas de los que adolece la selección de características es que, a pesar de ayudar a extraer un conocimiento más transparente y explicable acerca de las variables que son relevantes para el problema que estemos tratando, lo hace de una manera global. Esto es, no provee información caso por caso. Veamos un ejemplo: supongamos que tenemos un conjunto de datos de pacientes, y nos interesa predecir si un paciente concreto es propenso a padecer cáncer o no. Los métodos de selección de características son capaces de indicarnos cuáles son las características que más influyen a la hora de realizar dicha clasificación. Pero supongamos que tenemos un paciente, y queremos saber, exactamente, cuáles son las características que le han indicado al sistema que dicho paciente es propenso a padecer cáncer. Los sistemas clásicos de selección de características son incapaces de proveer dicha información.

En este trabajo proponemos abordar el problema de selección de características partiendo de este escenario . Nuestra idea propone la utilización de técnicas que nos indiquen, caso por caso, cuáles son las características de cada uno con la información más discriminante. Una vez detectadas, creare-

mos un algoritmo de selección de características, incluyendo aquéllas con un valor medio de discriminación más alto.

El resto del artículo estará estructurado de la siguiente manera: la sección II explicará el método que vamos a utilizar para la evaluación de la importancia de cada característica; la sección III propondrá nuestro algoritmo de selección de características, basado en redes neuronales; finalmente, la sección IV ofrecerá resultados experimentales sobre un conjundo de datasets públicos, y la sección V propondrá nuestras conclusiones y trabajos futuros.

II. Selección de características en Deep Learning

La literatura científica en el campo de métodos de selección de características que hagan uso de modelos de redes neuronales masivas, tales como por ejemplo, las Redes Convolucionales (CNN), es muy escasa en la actualidad. Uno de los trabajos más interesantes es una variante del LASSO para ser utilizada en CNNs, llamada DeepLASSO [1]. El método introduce un factor multiplicativo a los valores de entrada (llamado *máscara*), sobre el que se aplica un factor de regularización equivalente al del método LASSO (norma 1). Los autores de [1] proponen diferentes regularizaciones (LASSO, elastic Net, etc.) sobre la misma estructura. Las restantes aproximaciones a la selección de características en Deep Learning se englobarían dentro del subconjunto de técnicas denominado como *saliency*.

II-A. Saliency

Saliency es una técnica que surge en la visión por computador, con el objetivo de evaluar la *calidad* de cada uno de los píxeles que componen una imagen. Clásicamente, las redes neuronales se han visto como una caja negra, en la que se obtiene cierta salida a partir de los datos de entrada, pero sin ningún tipo de explicación más o menos transparente sobre cómo se ha llegado a esa solución. Actualmente existen dos maneras de calcular esta información de salida: de una manera semi-supervisada, o totalmente supervisada.

Las primeras técnicas que se utilizaron eran del tipo semi-supervisado [2], [3]. Constan de una red que es entrenada como un clasificador (binario o multiclase). Una vez entrenada la red, y dada una imagen de entrada, se realiza una retropropagación desde la salida hasta la entrada, para averiguar cuáles han sido los píxeles que más han influído en la salida esperada.

Los modelos totalmente supervisados son más recientes [4], [5], y son entrenados de una manera distinta. En este caso, se hace uso de la llamada *segmentación semántica*, ya que la salida de la red tiene el mismo tamaño que la imagen de entrada. Además, para cada imagen sabemos cuáles son los píxeles importantes (por ejemplo, si estamos detectando cierto tipo de objeto, los píxeles importantes son aquellos que pertenecen al obejto en cuestión, mientras que el fondo es considerado irrelevante). El modelo entrena la red para que la salida concuerde con nuestra segmentación previa de píxeles importantes. Estos modelos también suelen ser conocidos como *modelos de atención* [6], [7] cuando se utiliza una Red Neuronal Recursiva (RNN) como último paso para evaluar la calidad de cada píxel.

Las técnicas totalmente supervisadas obtienen mejores resultados, pero tienen un hándicap importante: para entrenar el modelo se necesita saber, a priori, cuáles son las características relevantes de cada uno de los ejemplos de entrada. En el caso de imágenes suele ser sencillo, pero no siempre es posible obtener dicha información, puesto que no siempre se sabe cuáles son las características que contienen la información más discriminante. Por tanto, para nuestro modelo de selección de características, proponemos hacer uso de una técnica de saliency semi-supervisada.

II-B. Aproximación propuesta

Para nuestro modelo vamos a utilizar una generalización de la idea propuesta en [2]. Sea un conjunto de datos de entrada $\mathbf{X} \in \mathcal{R}^{N\times R}$, con N número de muestras y R número de características; y $\tilde{\mathbf{Y}} = f(\mathbf{X};\mathbf{\Theta}) \in \mathcal{R}^{N\times C}$ nuestro clasificador, que puede ser tanto un clasificador linear como una CNN de muchas capas. En este caso C es el número de clases a evaluar, y $\mathbf{\Theta}$ son los pesos del clasificador que necesitan ser ajustados en la etapa de entrenamiento. Por simplificación, asumimos que los valores de $f(\mathbf{X};\mathbf{\Theta})$ son el resultado de aplicar la función softmax, devolviendo la probabilidad de pertenencia a cada clase. Esto es,

$$\sum_{c=1}^{C} \tilde{y}_c^{(i)} = 1 \tag{1}$$

Para entrenar la red se minimizará una función de pérdida $\ell(\mathbf{\Theta}; f, \mathbf{X}, \mathbf{Y})$, donde $\mathbf{Y} \in \mathcal{R}^{N\times C}$ es la codificación binaria de la clase esperada (*one-hot encoding*). En nuestro caso, minimizaremos la función de entropía cruzada (*cross-entropy*), definida como

$$\ell(\mathbf{\Theta}; f, \mathbf{X}, \mathbf{Y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_c^{(i)} \log\left(f(\mathbf{x}^{(i)};\mathbf{\Theta})_c\right) \tag{2}$$

En el caso de nuestra aproximación, se puede utilizar en conjunto con la técnica DeepLASSO [1], en cuyo caso la función de coste se modificaría de la siguiente manera

$$\ell(\mathbf{\Theta}, \psi; f, \mathbf{X}, \mathbf{Y}) = \|\psi\|_1 - \frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_c^{(i)} \log\left(f(\psi * \mathbf{x}^{(i)};\mathbf{\Theta})_c\right), \tag{3}$$

donde ψ es el vector *máscara*.

La idea de saliency es similar a la que se utiliza para actualizar los pesos $\mathbf{\Theta}$, calculando su derivada con respecto a la función de coste, esto es

$$\mathbf{\Theta} \leftarrow \mathbf{\Theta} - \alpha\frac{\partial \ell}{\partial \Theta}, \tag{4}$$

siendo α el factor de aprendizaje.

En cuanto a su significado numérico, los valores altos en magnitud de la derivada indican que dicho peso está influyendo negativamente en el funcionamiento correcto del sistema, mientras que valores cercanos a 0 indican que el peso está bien ajustado. En el caso del saliency, nos interesaría que las

características con magnitudes altas sean las que contienen información más significativa, mientras que las cercanas a 0 sean características irrelevantes. Nuestra función de saliency se define como

$$\sigma(\mathbf{x}^{(i)}; \mathbf{\Theta}, f, y^{(i)}) = \left| \frac{\partial g(\mathbf{x}^{(i)}; \mathbf{\Theta}, f, y^{(i)})}{\partial \mathbf{x}^{(i)}} \right|, \quad (5)$$

esto es, nos interesa saber el gradiente de los ejemplos de entrada con respecto a una función, que llamaremos *función de ganancia* (g), que devuelve su valor máximo cuando la clasificación es perfecta ($\ell = 0$), y valores cercanos a 0 cuando el clasificador falla. En el caso de la función de coste de entropía cruzada que estamos utilizando (Eq. 2), definimos la función de ganancia como

$$g(\mathbf{x}^{(i)}; \mathbf{\Theta}, f, y^{(i)}) = -\beta y^{(i)} \log \Big(1 - f(\mathbf{x}^{(i)}; \mathbf{\Theta})\Big), \quad (6)$$

donde β es un hiperparámetro que controla el decaimiento de la función de ganancia. Por defecto, $\beta = 1$. Para evitar un gradiente infinito, podamos aquellos valores de $f(\mathbf{x}^{(i)}; \mathbf{\Theta}) = 1$. Por defecto,

$$f(\mathbf{x}^{(i)}; \mathbf{\Theta}) = \text{mín}\{1 - e^{-8}, \quad f(\mathbf{x}^{(i)}; \mathbf{\Theta})\} \quad (7)$$

Es importante remarcar que esta técnica es similar a la propuesta en [2]. Se diferencia en un único punto: mientras que el método en [2] es específico para un problema de clasificación, dado que descomponen la última capa de la red para aplicar su idea, nuestro modelo opera directamente como una función matemática de ganancia, lo que lo hace susceptible de poder ser utilizado en otro tipo de problemáticas, como los modelos de regresión. Otra ventaja de nuestra aproximación es que, dado que se aplica sobre una función de ganancia del clasificador, el gradiente es cercano a 0 cuando el clasificador se equivoca. Por tanto, el sistema nos dice que no hay características relevantes, haciéndolo robusto ante ejemplos que no son clasificados correctamente por la red.

III. SELECCIÓN DE CARACTERÍSTICAS USANDO SALIENCY

Nuestro método de selección de características tendrá un comportamiento de tipo *ranker*, es decir, devolverá un vector ordenado de todas las características en función de su importancia. El esquema de nuestra aproximación puede verse en el Algoritmo 1.

El algoritmo propuesto tiene dos hiper-parámetros: $\epsilon \geq 0$, como control de parada, y $1 \geq \gamma > 0$, que controla el porcentaje de características *vivas* que van a ser mantenidas para la siguiente iteración.

Comenzamos entrenando la red neuronal f con todos los datos de entrenamiento. Luego, por cada clase calculamos el saliency de cada uno de los ejemplos, para luego normalizar mediante la norma 1. De esta manera para cada clase tenemos la probabilidad de que cada una de las características sea relevante para la clasificación. Una vez obtenidas las probabilidades de todas las clases, se suman y se ordenan, obteniendo el ranking de importancia de las características.

Datos: $\mathbf{X}, \mathbf{Y}, f, \ell, \mathbf{\Theta}, \gamma, \epsilon$, *reps*
Resultado: ranking de características $\mathbf{r}$
$n_f \leftarrow R$;
$\mathbf{r} \leftarrow [1 \dots n_f]$;
mientras $n_f > \epsilon > 1$ **hacer**
  $\hat{\mathbf{X}} \leftarrow \mathbf{X}$;
  $\hat{\mathbf{X}}[:, \mathbf{r}[n_f + 1 : R]] \leftarrow 0$;
  $\sigma_{fs} \leftarrow \text{zeros}(n_f)$;
  para $r \leftarrow 1$ **a** *reps* **hacer**
    Inicializar $f(\hat{\mathbf{X}}; \mathbf{\Theta})$;
    Entrenar $f(\hat{\mathbf{X}}; \mathbf{\Theta})$ dado $\mathbf{Y}$;
    para $c \leftarrow 1$ **a** C **hacer**
      $\sigma_c \leftarrow \sum_{i_c=1}^{N_c} \sigma(\mathbf{x}^{(i_c)}; \ell, \mathbf{\Theta}, f, y^{(i_c)})$;
      $\sigma_{fs} \leftarrow \sigma_{fs} + \frac{\sigma_c}{\|\sigma_c\|_1}$;
    fin
  fin
  index $\leftarrow$ argsort(σ_{fs}, descend);
  $\mathbf{r}[1 : n_f] \leftarrow \mathbf{r}$[index];
  $n_f \leftarrow int(n_f * \gamma)$;
fin

Algoritmo 1: Selección de características usando saliency

No obstante, hay que tener en cuenta que las redes neuronales son propensas al *sobre-entrenamiento* (overfitting). Esto haría que, en caso de que se eliminaran ciertas características, el reentrenamiento de la red cambiaría sustancialmente los valores de los pesos de la misma. Por esta razón, utlizaremos dos variables: (a) *reps*, que entrenará la red un número determinado de veces, para evitar valores atípicos; y (b), una vez obtenido el ranking de características, utilizamos el hiper-parámetro γ para eliminar un porcentaje de las características, con objeto de volver a reentrenar la red y ajustar el orden de las características áun activas. El algoritmo se detendrá cuando el número de características activas sea inferior al hiper-parámetro ϵ.

Por lo tanto, la complejidad del algoritmo es variable, pues depende completamente del parámetro γ. En los casos extremos, tenemos que si $\gamma = 0$, entonces la complejidad es lineal ($\mathcal{O}(Nreps)$), dependiendo sólo del número de ejemplos y de repeticiones. Sin embargo, con un $\gamma \approx 1$, la complejidad pasa a depender del número de variables ($\mathcal{O}(RNreps)$), pues necesitamos entrenar tantas redes como características distintas tengamos.

IV. RESULTADOS EXPERIMENTALES

Para evaluar el comportamiento de nuestra aproximación, hemos utilizado los 5 datasets propuestos en el *NIPS 2003 Features Selection challenge* [1]. Se trata de una serie de datasets sintéticos, diseñados con el objetivo de medir la calidad de los resultados obtenidos por los métodos de selección de características. Las características específicas de cada uno de estos datasets se muestran en la tabla I. Es un conjunto

[1] http://clopinet.com/isabelle/Projects/NIPS2003/

Cuadro I
DATASETS UTILIZADOS.

Conjunto	# ejemplos (entr., test)	# total características	# características válidas	% características válidas	ratio positivos/negativos
Arcene	(88, 112)	10000	7000	0.7	1.0
Dexter	(300, 300)	20000	9947	0.5	1.0
Dorothea	(800, 350)	100000	50000	0.5	0.11
Gisette	(6000, 1000)	5000	2500	0.5	1.0
Madelon	(2000, 600)	500	20	0.04	1.0

de datasets particularmente complejo porque incluye diversos tipos de casuísticas: pocos ejemplos de entrenamiento (Arcene), datos desbalanceados (Dorothea) o pocas características válidas (Madelon).

El algoritmo propuesto tiene la ventaja de que es capaz de calcular el ranking de las características al mismo tiempo que entrena el clasificador. Por tanto, se podría pensar que el subconjunto de características seleccionado es dependiente de la arquitectura. Por este motivo, hemos decidido separar la obtención de características del entrenamiento de la red, usando dos arquitecturas distintas para cada cometido. Con este punto creemos poder realizar una comparación más justa con otros métodos que no poseen esta característica, como son los basados en el análisis de información mutua [8].

IV-0a. Arquitectura para la obtención del ranking de características: Para este cometido hemos decidido utilizar una red neuronal totalmente conectada con 3 capas ocultas de 150, 100 y 50 elementos, respectivamente. Utilizamos *Batch Normalization* (BN) [9] seguido de la activación $ReLU(x) = max(0, x)$ en cada capa; como salida usamos un softmax, y utilizamos la entropía cruzada (Eq. 2) como función de coste.

Asimismo, usamos un *weight decay* de 0,001 para todos los pesos, con el objeto de eliminar el sobreentrenamiento. Entrenamos dicha red para un total de 100 iteraciones sobre el conjunto de entrenamiento, usando Adam [10] como optimizador. En el caso de tener datos desbalanceados, aumentamos el número de muestras hasta igualar los ejemplos disponibles para cada clase.

Para la creación y entrenamiento de esta red se ha utilizado el framework Keras [2] sobre Tensorflow [11].

IV-0b. Arquitectura del clasificador: Dado el pequeño número de muestras que tenemos en los conjuntos, nos hemos decantado por utilizar como clasificador una *Máquina de vector soporte* (SVM) con kernel gaussiano (RBF). Hemos usado el hiper-parámetro $C = 1$, sobre la implementación existente en la librería *scikit-learn* de Python.

IV-A. Efecto del parámetro γ

En primer lugar queremos conocer cuál es el efecto que producen los hiper-parámetros γ y *reps* (ver sección III y algoritmo 1) . En el caso del parámetro γ, hemos ejecutado nuestro algoritmo utilizando para ello sólo una repetición para cada conjunto de características (*reps* $= 1$). En la Fig. 2 podemos observar cómo la precisión del algoritmo mejora conforme vamos aumentando el valor de γ. Esto es debido a que, dado que el número de ejemplos de cada dataset es reducido, se produce mucho sobre-entrenamiento en la red. Dicho sobre-entrenamiento causa que determinadas características pasen a ser relevantes, únicamente debido a la inicialización del modelo.

Como caso inusual, en el dataset *Dorothea* se produce el efecto contrario. Esto puede ser debido tanto a la naturaleza de las características (binarias) del dataset, como a lo desbalanceado de los datos (muchos más ejemplos negativos que positivos).

IV-B. Efecto del parámetro reps

Como comentamos anteriormente, el sobre-entrenamiento introduce una variabilidad indeseada en la valoración de las características. Así como el uso del hiper-parámetro γ es una manera de tratar de solventar este inconveniente, éste también podría ser mitigado aumentando el número de veces que entrenamos la red, esto es, aumentando el parámetro *reps*.

Por tanto, hemos ejecutado nuestro algoritmo, fijando para ello el parámetro $\gamma = 0$, y variando el número de repeticiones. La Fig. 3 muestra los resultados obtenidos. Contrariamente a lo ocurrido con el hiper-parámetro γ, el aumento del número de repeticiones no conlleva una mejora sustancial de los resultados obtenidos. De igual manera, en el dataset *Dorothea* podemos ver que se produce el mismo efecto que ya habíamos presenciado con el hiper-parámetro γ.

IV-C. Evaluación de rendimiento

Para probar el rendimiento de nuestra propuesta, hemos decidido comparar nuestro método con una serie de algoritmos clásicos de selección de características: *MIM* [12] y *ReliefF* [13], en sus implementaciones existentes en la librería Weka [14]. Pese a que son algoritmos ya clásicos, siguen siendo utilizados en la actualidad por su probada eficacia. Los hemos escogido por ser métodos que devuelven un ranking de características, de la misma manera que lo hace nuestra propuesta. Además de estos algoritmos en Weka, hemos preparado también una variación del DeepLASSO [1] para poder usarlo en el estudio comparativo. El funcionamiento de éste es equivalente al de nuestro algoritmo, con la salvedad de que en lugar de una función de saliency, utilizaremos el valor absoluto de los valores de la máscara γ descrita en Eq. 3 como la medida de calidad de las características.

Como hemos mencionado anteriormente, hemos decidido separar la obtención del subconjunto de características relevantes del entrenamiento del clasificador, usando dos arquitecturas distintas para cada cometido. Con este punto creemos poder

[2]https://keras.io/

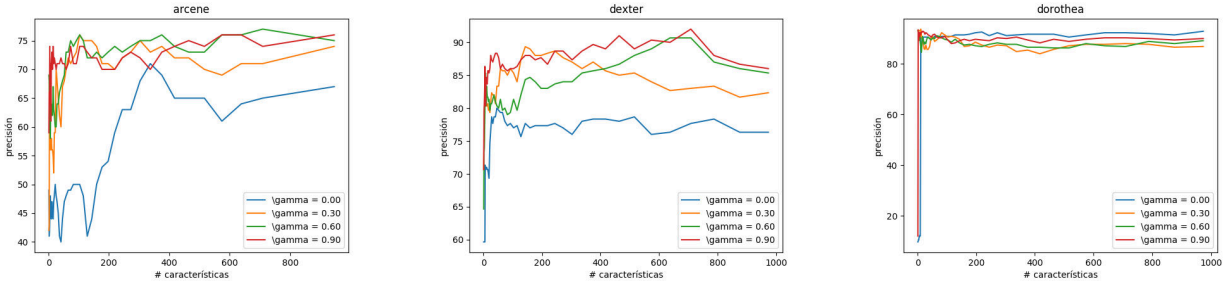


Figura 2. Efecto del hiper-parámetro γ sobre el algoritmo. Conforme su valor se acerca a 1, la selección de características se vuelve más precisa, especialmente cuando el número de características utilizadas es reducido.

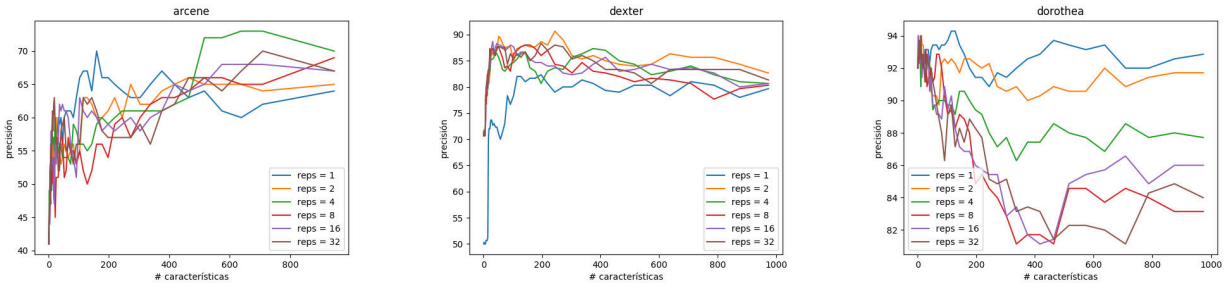


Figura 3. Efecto del hiper-parámetro *reps* sobre el algoritmo. Al contrario que con γ, su aumento no parece suponer una mejora significativa en los resultados.

realizar una comparación más justa contra los métodos *MIN* y *ReliefF*.

En la tabla II podemos observar los resultados obtenidos. En general, nuestra aproximación funciona de una manera análoga a los algoritmos basados en información mutua, algo remarcable dado que estamos utilizando dos arquitecturas distintas para el cálculo del ranking y la clasificación. Podemos observar como la técnica de DeepLASSO no obtiene buenos resultados cuando no utilizamos la misma arquitectura tanto para ranking como para la clasificación. Asimismo, también podemos observar como, en aquellos datasets que contienen más ejemplos (Gisette y Madelon), nuestro método es el aue ofrece los mejores resultados. Asimismo, el valor del parámetro *Área bajo la curva* (AUC) de nuestra propuesta es mayor es prácticamente todos los conjuntos, lo que sugiere una robustez mayor en la selección de características.

Analizando un ranking de las posiciones medias en las que se situarían los cuatro métodos comparados, veríamos que nuestra propuesta es el que tiene mejor media en cuanto a precisión, con un puesto 1,6 de media entre los cuatro métodos, y es además es el segundo de de los métodos, a muy poca diferencia de DeepLasso, que consigue mejores resultados con un subconjunto de características seleccionado más bajo (véase la tabla III). Nótese que la diferencia es bastante importante en conjuntos como Dexter, en los que consigue la segunda mejor precisión (con una diferencia de 0,04, pero con 28 características en vez de las 103 del mejor método en precisión, que en este caso es MIM). Así pues nuestra propuesta es bastante estable, consguiendo los mejores resultados de precisión en media, con un número de características bajo.

V. Conclusiones y Trabajo Futuro

En este paper hemos propuesto un nuevo algoritmo de selección de características basado en *saliency*. Dado un clasificador, y la creación de una función de ganancia, es posible conocer, para cada ejemplo, cuáles son las características más importantes a la hora de clasificar su comportamiento. Esta propiedad, por sí sola, añade una capacidad a nuestro algoritmo de la que carecen los métodos clásicos de selección de características. Asimismo, se trata de un método muy flexible, puesto que permite su uso en casi cualquier clasificador usado en la actualidad (SVM, CNN, ...). Por tanto, se trata de una herramienta muy útil para la selección de características en entornos de Big Data.

Como trabajo futuro, planteamos dos alternativas diferentes. En primer lugar, probar nuestro algoritmo con Big Data, especialmente con CNNs y datasets mucho más grandes, con objeto de confirmar los resultados obtenidos en este paper. Es de esperar que los resultados sean aún mejores, dado que el entrenamiento de la red sería aún más preciso. En segundo lugar, planteamos la modificación del Algoritmo 1, probando distintas configuraciones para el cálculo de σ_{fs}, tales como el uso de técnicas de ranking, o la eliminación de normalizaciones.

Referencias

[1] Y. Li, C.-Y. Chen y W. W. Wasserman, "Deep feature selection: theory and application to identify enhancers and promoters", *Journal of Computational Biology*, vol. 23, n.º 5, págs. 322-336, 2016.

Cuadro II

Resultados obtenidos. Dado que los datasets contienen características artificiales, sólo evaluamos su rendimiento hasta un número de características, como máximo, igual al número de características reales (válidas) de cada dataset.

Conjto.	# carac. válidas	Método	Mejor Precisión (Nº carac.)	Prec. 10 % carac. válidas	Prec. 25 % carac. válidas	Prec. 50 % carac. válidas	Prec. 100 % carac. válidas	AUC
Arcene	7000	MIM	81.0 (337)	75.0	74.0	74.0	73.0	0.736
		ReliefF	**83.0** (375)	**77.8**	**80.0**	**80.0**	71.0	**0.786**
		DeepLASSO	80.0 (464)	73.0	69.0	71.0	70.0	0.709
		Propuesta	81.0 (464)	76.0	72.0	77.0	**80.0**	0.773
Dexter	9947	MIM	**90.7** (103)	73.3	59.3	51.3	50.0	0.567
		ReliefF	86.0 (1204)	**85.0**	59.7	49.7	50.0	0.569
		DeepLASSO	87.3 (9)	62.3	53.0	49.7	50.0	0.533
		Propuesta	90.3 (28)	83.7	**77.0**	**72.0**	**52.0**	**0.732**
Dorothea	2500	MIM	94.3 (5)	88.0	81.7	63.1	**90.6**	0.794
		ReliefF	**94.6** (876)	**92.9**	**92.0**	23.4	**90.6**	0.682
		DeepLASSO	94.3 (5)	37.7	90.9	**90.3**	90.3	0.765
		Propuesta	94.3 (3)	90.9	89.7	88.9	89.1	**0.895**
Gisette	2500	MIM	98.4 (689)	97.0	98.1	**98.4**	97.9	0.978
		ReliefF	98.4 (1226)	97.7	98.0	**98.4**	**98.3**	0.980
		DeepLASSO	97.4 (82)	96.1	96.0	95.9	97.3	0.964
		Propuesta	**98.5** (516)	**98.0**	**98.5**	98.1	**98.3**	**0.981**
Madelon	2500	MIM	84.8 (13)	**67.8**	72.7	80.7	78.7	0.777
		ReliefF	85.3 (19)	62.7	61.8	80.7	**85.3**	0.754
		DeepLASSO	86.7 (6)	58.8	**86.0**	77.8	75.5	0.755
		Propuesta	**87.0** (6)	59.5	85.8	**86.2**	**85.3**	**0.828**

Cuadro III

Ranking de los algoritmos en cuanto a precisión, y en cuanto al número más bajo de características usadas (entre paréntesis)

Método	Posición Arcene	Pos. Dexter	Pos. Dorothea	Pos. Gisette	Pos. Madelon	Posición Media
ReliefF	1 (2)	4 (4)	2 (4)	2 (4)	3 (3)	2,4 (3,4)
Propuesta	2 (3)	2 (2)	2 (1)	1 (2)	1 (1)	**1,6** (1,8)
MIM	2 (1)	1 (3)	2 (3)	2 (3)	4 (2)	2,2 (2,4)
DeepLasso	3 (3)	3 (1)	2 (2)	3 (1)	2 (1)	2,6 (**1,6**)

[2] K. Simonyan, A. Vedaldi y A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps", *arXiv preprint arXiv:1312.6034*, 2013.

[3] A. Mahendran y A. Vedaldi, "Understanding deep image representations by inverting them", en *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, IEEE, 2015, págs. 5188-5196.

[4] R. Zhao, W. Ouyang, H. Li y X. Wang, "Saliency detection by multi-context deep learning", en *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, págs. 1265-1274.

[5] D. Zhang, D. Meng y J. Han, "Co-saliency detection via a self-paced multiple-instance learning framework", *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, n.º 5, págs. 865-878, 2017.

[6] V. Mnih, N. Heess, A. Graves y col., "Recurrent models of visual attention", en *Advances in neural information processing systems*, 2014, págs. 2204-2212.

[7] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel e Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention", en *International Conference on Machine Learning*, 2015, págs. 2048-2057.

[8] V. Bolón-Canedo, N. Sánchez-Maroño y A. Alonso-Betanzos, "A review of feature selection methods on synthetic data", *Knowledge and information systems*, vol. 34, n.º 3, págs. 483-519, 2013.

[9] S. Ioffe y C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", *arXiv preprint arXiv:1502.03167*, 2015.

[10] D. P. Kingma y J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[11] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard y col., "TensorFlow: A System for Large-Scale Machine Learning.", en *OSDI*, vol. 16, 2016, págs. 265-283.

[12] M. A. Hall y L. A. Smith, "Practical feature subset selection for machine learning", 1998.

[13] I. Kononenko, "Estimating attributes: analysis and extensions of RELIEF", en *European conference on machine learning*, Springer, 1994, págs. 171-182.

[14] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann e I. H. Witten, "The WEKA data mining software: an update", *ACM SIGKDD explorations newsletter*, vol. 11, n.º 1, págs. 10-18, 2009.