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.

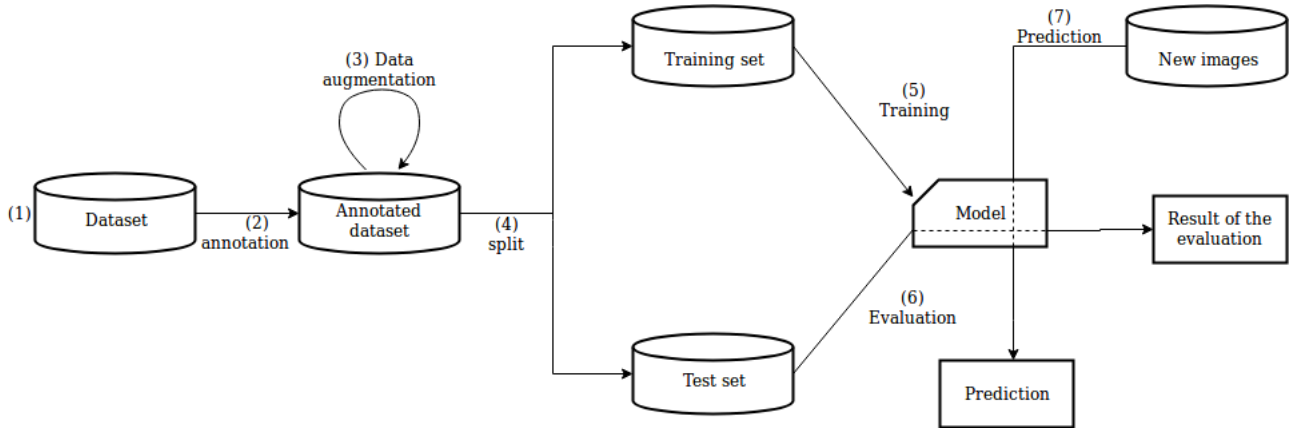


Fig. 1. Workflow of object detection projects

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

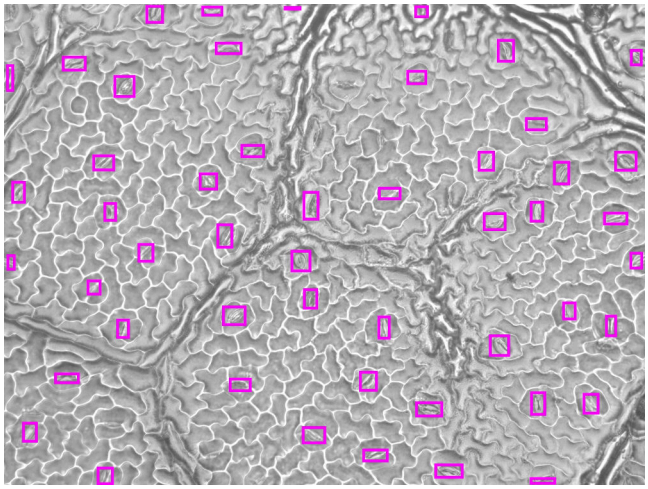


Fig. 2. Stomata identification results

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:

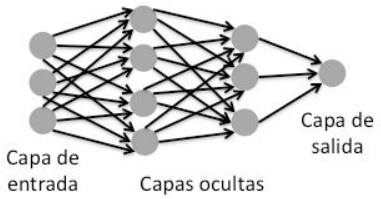
- Los modelos DL incorporan mecanismos para generar nuevas características por sí mismos, sin necesidad de realizar esto en fases externas.
- Las técnicas DL mejoran el rendimiento en cuanto a tiempo de cómputo, al realizar algunas de las tareas más costosas, como la generación de nuevas características.
- 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.

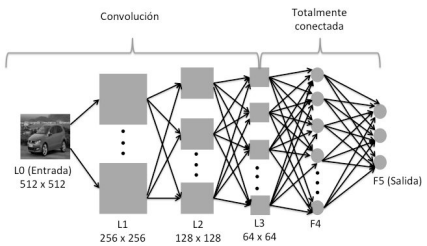
A. Redes Neuronales Convolucionales

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].

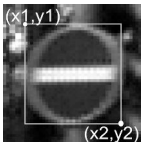


Fig. 2. Ejemplo de imagen recortada (fuente: http://benchmark.ini.rub.de).

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.

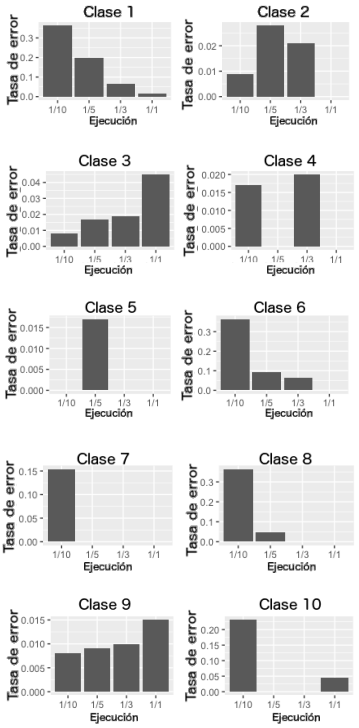


Fig. 3. Tasa de Error por clase y experimento.

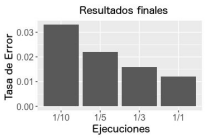


Fig. 4. Tasa media de Error por experimento.

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:

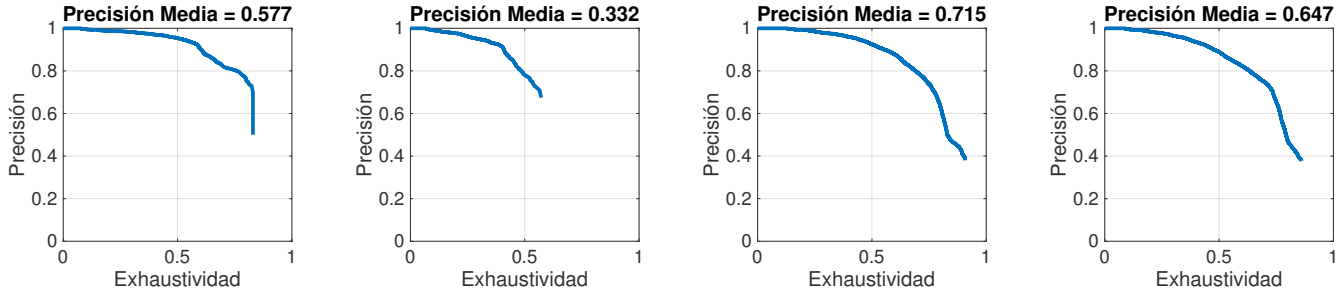
$$a\_gt_m < u \cdot (height \cdot width) \quad (4)$$

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)

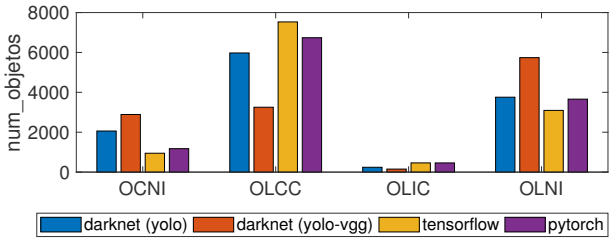


Figura 2: Comparativa de rendimiento de la clasificación de objetos. El número de objetos clasificados por marco de trabajo se muestra de acuerdo a las condiciones consideradas: Objetos Localizados Correctamente Clasificados (OLCC), Objetos Localizados Incorrectamente Clasificados (OLIC), Objetos Cercanos 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.

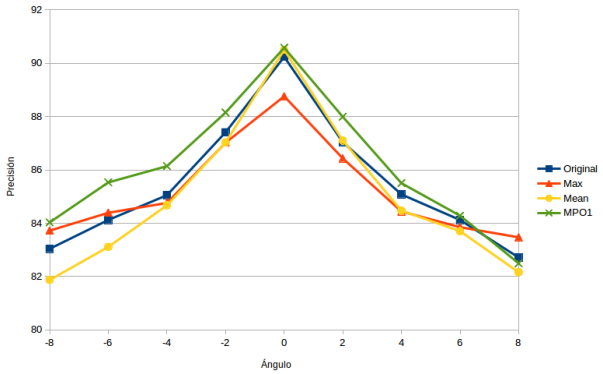


Figura 1. Test de Robustez giro NiN en CIFAR-10.

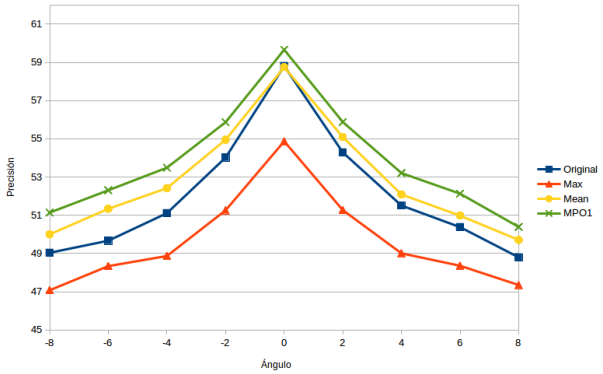


Figura 2. Test de Robustez giro NiN en CIFAR-100.

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

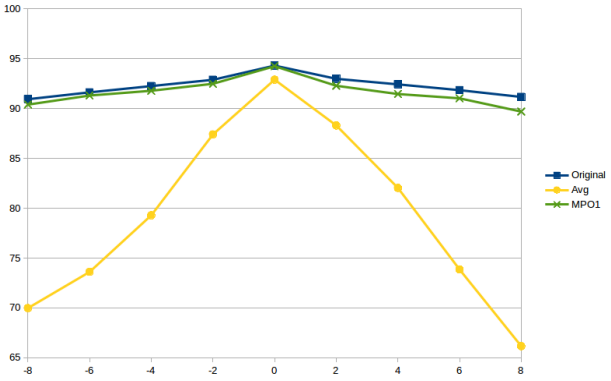


Figura 3. Test Robustez giro VGG en FMNIST.

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.

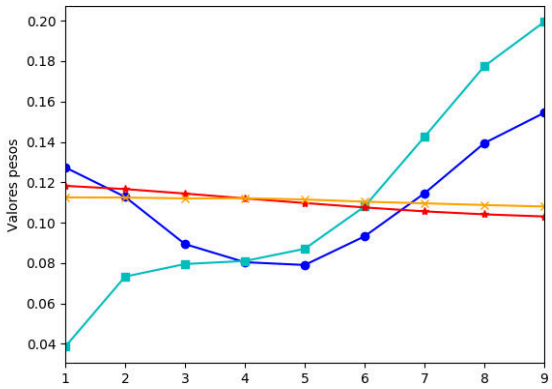


Figura 4. Azul: pesos de la capa pool1 entrenando en CIFAR-10; Cyan pesos de la capa pool2 entrenando en CIFAR-10; Rojo: pesos de la capa pool1 entrenando en CIFAR-100; Naranja pesos de la capa pool2 entrenando en CIFAR-100.

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).

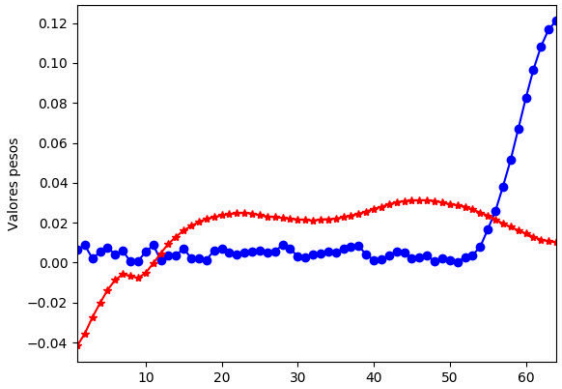


Figura 5. Azul: pesos de la capa pool3 entrenando en CIFAR-10; Rojo: pesos de la capa pool3 entrenando en CIFAR-100.

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.

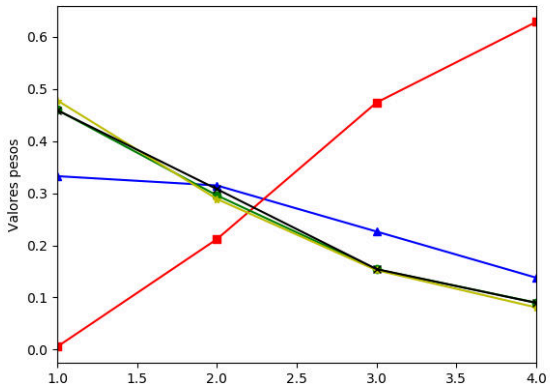


Figura 6. Pesos de las capas pool1 (en azul), pool2 (en rojo), pool3 (en verde), pool4 (en amarillo) y pool5 (en negro) de la red VGG (MPO1) para FMNIST.

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'.

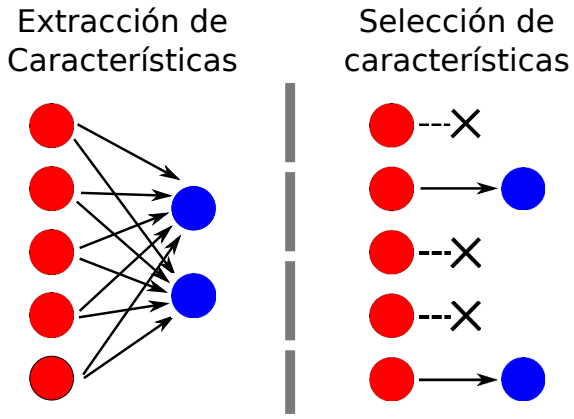


Figura 1. La extracción de características crea nuevas características mediante la combinación del conjunto inicial, mientras que la selección de características elimina aquellas que considera irrelevantes o redundantes, manteniendo inalteradas las restantes.

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/

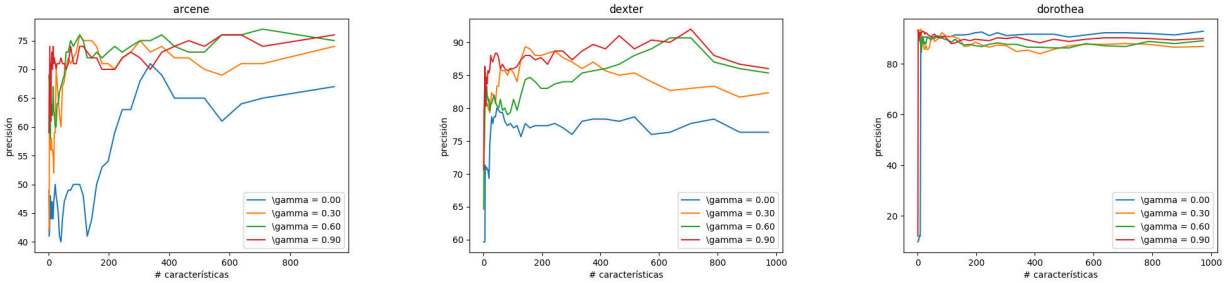


Figura 2. Efecto del hiper-parámetro γ sobre el algoritmo. Conforme su valor se acerca a 1, la selección de características se vuelve más precisa, especialmente cuando el número de características utilizadas es reducido.

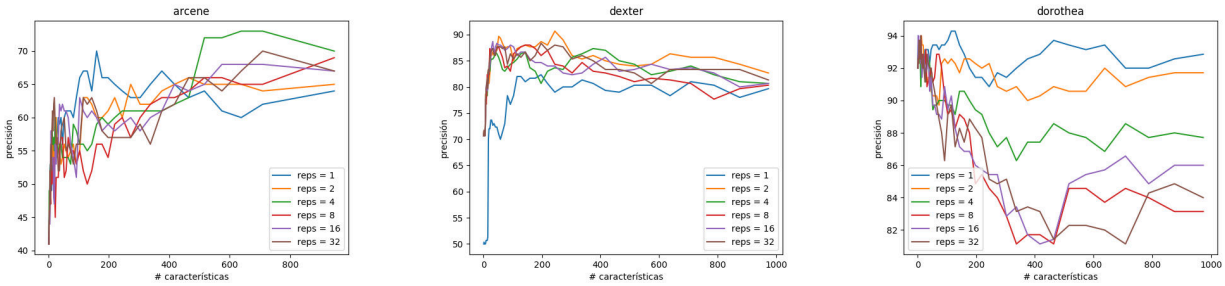


Figura 3. Efecto del hiper-parámetro *reps* sobre el algoritmo. Al contrario que con γ, su aumento no parece suponer una mejora significativa en los resultados.

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.