Facultad
Informática
Universidad
Murcia

Universidad de Murcia
Facultad de Informática

GRADO EN INGENIERÍA INFORMÁTICA

Aplicación de técnicas de *Deep Learning* usando el framework MXNet en un caso médico real

---

Trabajo de Fin de Grado realizado por:
Eduardo José Gómez Hernández
{eduardojose.gomez@um.es}
Bajo la dirección de:
José Manuel García Carrasco
{jmgarcia@ditec.um.es}
Junio, 2018

Esta página ha sido intencionadamente dejada en blanco

Declaración firmada sobre la originalidad del trabajo

D./Dña. Eduardo José Gómez Hernández, con DNI 23304545W, estudiante de la titulación de Grado de Ingeniería Informática de la Universidad de Murcia y autor del TF titulado “Aplicación de técnicas de Deep Learning usando el framework MXNet en un caso médico real”.

De acuerdo con el Reglamento por el que se regulan los Trabajos Fin de Grado y de Fin de Máster en la Universidad de Murcia (aprobado C. de Gob. 30-04-2015 y modificado 22-04-2016), así como la normativa interna para la oferta, asignación, elaboración y defensa de los Trabajos Fin de Grado y Fin de Máster de las titulaciones impartidas en la Facultad de Informática de la Universidad de Murcia (aprobada en Junta de Facultad 27-11-2015)

DECLARO:

Que el Trabajo Fin de Grado presentado para su evaluación es original y de elaboración personal. Todas las fuentes utilizadas han sido debidamente citadas. Así mismo, declara que no incumple ningún contrato de confidencialidad, ni viola ningún derecho de propiedad intelectual e industrial

Murcia, a 24 de Junio de 2018

Fdo.: Eduardo José Gómez Hernández
Autor del TF

Esta página ha sido intencionadamente dejada en blanco

Índice

Resumen

Desde los últimos años, el campo del Aprendizaje Computacional se encuentra en auge, sobre todo debido a las técnicas de *Deep Learning* (Aprendizaje Profundo). Este campo engloba un conjunto de técnicas, la mayoría desarrolladas en el siglo XX entre los años 50 y 60, en las que se requiere de un proceso de aprendizaje. Las primeras redes neuronales eran ADALINE y MADALINE, las cuales se encargaban de predecir el siguiente bit en la red telefónica y de reducir el eco en estas, respectivamente. El proceso de aprendizaje puede ser realizado de múltiples maneras y conlleva varios días o semanas, tiempo que puede ser reducido con el uso de HPC (*High-Performance Computing* o Computación de Alto Rendimiento).

Las redes neuronales están formadas por unidades básicas llamadas neuronas (perceptrones), las cuales imitan el comportamiento de las neuronas humanas, teniendo un conjunto de entradas que simulan las dendritas, una salida que simula el axón, y una unidad funcional que simula el procesamiento de la neurona. Por sí solas las neuronas no son capaces de solucionar problemas complejos, por lo que para ello se agrupan en capas. Estas capas conectan sus salidas con las entradas de la siguiente capa para construir una red neuronal.

Al unir tres o más capas, se forman las redes neuronales profundas, en caso de estar formadas únicamente por perceptrones se las conoce como MLP (*MultiLayer Perceptron* o Perceptrones Multi Capa). Las MLP son capaces de representar cualquier tipo de función matemática, sin embargo debido a la falta de cómputo para algunos problemas, se introdujeron algunos nuevos tipos de capas y metacapas.

Uno de los tipos de capas que se han creado es la capa de convolución, la cual está destinada a trabajar con imágenes aplicando filtros; otra es la capa de *pooling*, que permite reducir el tamaño de entrada. A la red formada por multiples capas de convolución y *pooling* unidas a una MLP se la conoce como Red Neuronal de Convolución. En este tipo de red se puede observar algunas capas que tienen un funcionamiento más indirecto, como por ejemplo la capa *flattern* que se encarga de estructurar los datos como un solo *array* unidimensional, permitiendo pasar los datos desde las capas de convolución y *pooling* a las de perceptrones.

Existen redes neuronales formadas únicamente por capas convolucionales, por ejemplo la estructura *Encoder-Decoder*, la cual se utiliza en el reconocimiento de figuras en imágenes. Estas redes están formadas por las capas habituales de una red de convolución pero añadiendo las capas de deconvolución y muestreo, que realizan el proceso inverso a su homónimas de convolución y *pooling* respectivamente.

Para facilitar la tarea de creación y entrenamiento de estas redes, se desarrollaron varios conjuntos de herramientas llamados *frameworks*. Los *frameworks* más conocidos son TensorFlow, Keras y Theano, aunque para el presente trabajo la herramienta elegida es MXNet. El *framework* elegido está bajo el proyecto Apache Incubator, el cual otorga fondos y recursos a aquellos proyectos que lo necesiten. Su evolución ha sido tan importante que empresas como Intel o Amazon, entre otras, están dedicando recursos para su desarrollo, ya sea mediante aceleradores hardware como FPGAs, o soporte software como Intel OpenVino.

En las redes neuronales, la fase que más tiempo de cómputo consume es la fase de entrenamiento, la cual realiza una búsqueda heurística para solucionar un problema de optimización. Existen varias maneras de entrenar la red, siendo las más conocidas el aprendizaje supervisado y el aprendizaje no supervisado.

En el aprendizaje supervisado el algoritmo más conocido es el SDG (*Stochastic Gradient Descendent* o Gradiente Descendiente Estocástico), una variación del Gradiente Descendiente aplicado en lotes. El funcionamiento de este algoritmo es obtener un mínimo global comparando la salida obtenida con lo que se espera obtener, y propaga el error en cada iteración.

El aprendizaje no supervisado, en cambio, funciona mediante el refuerzo de los enlaces de las neuronas activadas en cada iteración del algoritmo. Así, al no otorgar a la red del resultado que debe obtener, esta solo puede extraer información de las entradas y obtener el patrón a partir de ellas.

El rendimiento de un modelo de red neuronal se mide o bien usando el porcentaje de acierto de

la red (*accuracy*), o bien mediante el error acumulado (*loss*). Además, existen otras variables que modifican cómo se realiza el aprendizaje de la red; ejemplos son el tamaño de lote, el número de épocas, la *learning rate*, entre otras.

Para comprobar el correcto funcionamiento del *framework* se han usado: las bases de datos MNIST y CIFAR. MNIST es una base de datos que contiene números manuscritos digitalizados en cuadrados de 28x28, mientras que CIFAR es una base de datos de imágenes clasificadas en categorias. Tras la instalación del *framework*, se ha usado dichas bases de datos con las configuraciones recomendadas del *framework* para comprobar su correcto funcionamiento, obteniendo un 98.1 % y un 93.1 % de *accuracy* respectivamente.

Habiendo *jugado* un poco con el *framework*, llega el momento de aplicarlo a un caso real. Debido a la importancia social de la aplicación de estas técnicas en medicina, hemos decidido realizar un caso médico, en concreto con la colaboración de Hospital Reina Sofia de Murcia y el Dr. Enrique Poblet Martinez.

Hasta ahora estas técnicas eran usadas en imágenes radiológicas, sin embargo desde el estudio del cancer de mama usando técnicas de deep learning se está aplicando a otros tipos de imágenes médicas. Tal es la importancia que varios hospitales del mundo han empezado a digitalizar una gran cantidad de muestras de todo tipo para su posterior procesamiento. El objetivo de la aplicación de estas técnicas es conseguir reducir los tiempos de diagnóstico aumentando la eficacia de estos.

Con el Dr. Enrique Poblet, hemos decido estudiar el caso de la detección de zonas cancerígenas en biopsias de próstata, en concreto a modo de *screening* para poder priorizar aquellos casos que lo requieran, y otorgar una ayuda de la localización de dichas zonas cancerígenas.

No se le suele dar importancia, pero hay una etapa muy importante, el preprocesado de las imágenes de entrada. En este caso se ha decidido por el troceado de las imágenes, como se ha empezado a usar en otro tipo de biopsias empezando a dar un buen resultado.

Las biopsias son escaneadas a multiples aumentos y almacenadas en un fichero Tiff para su procesamiento. Usando las herramientas de la suite de ImageMagick se ha extraido la capa de 20 aumentos en Jpeg. Estas imágenes tienen dimensiones cercanas a los 5 GigaPíxeles (color RGB y 3 bytes por píxel), por lo que requieren de algún tipo de procesado para ser manejables. En este caso se ha optado por el cortado la imagen en *tiles* de tamaño 128x128 píxeles, ya que esta técnica resultó viable en la detección del cáncer de mama.

Aún así es necesario realizar algún tipo de preprocesamiento que reduzca la cantidad de variables de entrada, ya que a mayor cantidad de variables más complicado resultado el entrenamiento, en este caso hemos usado capas de convolución y el troceado en *tiles*. Posteriormente los *tiles* han sido etiquetados mediante la inferencia de la imagen original con la imagen.

Tras las pruebas realizadas, se puede observar que con el debido estudio podría obtenerse un modelo con suficiente precisión como para ser usado de *screening* en situaciones reales.

En el entrenamiento se ha tenido que generar un conjunto de entrenamiento y uno de validación, este conjunto ha tenido que ser repartido teniendo en cuenta que el objetivo de la red es diferenciar las zonas cancerígenas de las que no lo son, y estableciendo un 80 % de datos para el entrenamiento y un 20 % para la validación.

A un médico no se le puede dar un conjunto de 0s y 1s, sino una visión más fácil de entender y usar, por lo que se realiza un proceso que permita reconstruir la imagen de la biopsia, dejando marcadas aquellas zonas con una alta probabilidad de contener cáncer.

Es necesario que la cantidad de falsos negativos sea 0, ya que un paciente podría quedarse sin diagnóstico. Sin embargo aunque el falso positivo no es bueno, el médico corrigiría el diagnóstico en caso de ser necesario.

En este trabajo hemos conseguido obtener un 78.1 % de *accuracy*, el cual podría ser incrementado con el estudio necesario. Además hemos conseguido generar una metodología para seguir el estudio en este caso en concreto.

Extended Abstract

For the last few years, the field of Computational Learning han been booming, especially due Deep Learning techniques. This field encompasses a set of techniques, most of wich developed in the twentieth century between the 50s and 60s, in which a learning process is required. The first neural networks were ADALINE and MADALINE, which were responsible for predicting the next bit in the telephone network and reducing the echo in these, respectively. The learning process can be performed in multiple ways and may take several days or weeks, a time that can be reduced with the use of HPC (High-Performance Computing).

The Neural networks are formed by basic units called neurons (perceptrons), which mimic the behaviour of human neurons, as they have a set of inputs that simulate dendrites, an output that simulates the axon, and a functional unit that simulates the processing of the neuron. On their own, neurons are unable to solve complex problems, so they group in layers. For that these layers connect their outputs with the inputs of the next layer thus building a neural network.

When three or more layers bind, deep neural networks are formed. If they are formed only by perceptrons they are refered to as as MLP (MultiLayer Perceptron). MLPs are able to represent any type of mathematical function, however due to lack of computation for some problems, new types of layers and metalayers were introduced.

One of the layer types that were created is the convolution layer, whose purpose is to work with images by applying filters; another is the pooling layer, which allows for the reduction of the input size. The network formed by multiple layers of convolution and pooling linked to an MLP is known as a Neural Convolution Network. In this type of network some layers that have a more indirect operation can be observed, such as the flattern layer that is responsible for structuring the data as a single one-dimensional array, thus allowing the data to pass from the convolution and pooling layers to those of perceptrons

There are neural networks that are formed only by convolutional layers, an example is the Encoder-Decoder structure, which is used in the recognition of figures in images. These networks are formed by the usual layers of a convolution network but also adds the deconvolution and sampling layers, which perform the reverse process in comparison with their convolution and pooling homonyms.

To facilitate the task of creating and training these networks, several sets of tools called frameworks have been developed. The most known frameworks are TensorFlow, Keras and Theano, although for the present work the chosen tool has been MXNet. The chosen framework is under the Apache Incubator project, which grants funds and resources to those projects that are the weed for then it. Its evolution has been of such important that companies such as Intel or Amazon, among others, are devoting resources to its development, either through hardware accelerators such as FPGAs, or software support such as Intel OpenVino.

In neural networks, the phase that consumes the most computing time is the training phase, which performs a heuristic search to solve an optimization problem. There are several ways to train the network, the best known being supervised learning and unsupervised learning.

In supervised learning the best-known algorithm is SDG (Stochastic Descendant Gradient), a variation of Descendant Gradient applied in batches. The operation of this algorithm is to obtain a global minimum comparing the output achieved with what is expected, and to propagate the error in each iteration.

The unsupervised learning, on the other hand, works by reinforcing the links of the activated neurons in each iteration of the algorithm. Thus, by not granting the network the result it must obtain, it can only extract information from the entries and obtain the pattern from them.

The performance of a neural network model is measured either by using the accuracy percentage of the network (accuracy), or by the accumulated error (loss). Besides, there are other variables that modify how network learning is carried out; examples of this are batch size, number of times, learning rate, among others.

To verify the correct functioning of the framework the MNIST and CIFAR databases have been

used. MNIST is a database that contains handwritten numbers digitized in 28x28 squares, while CIFAR is a database of images classified into categories. After the installation of the framework, these databases have been used with the recommended configurations of the framework to verify its correct functioning, obtaining a 98.1 % and a 93.1 % accuracy respectively.

Having already played slightly with the framework, it was time to apply it to a real case. Due to the social importance of the application of these techniques in medicine, we have decided to test it in a medical case, specifically with the collaboration of Hospital Reina Sofía of Murcia and Dr Enrique Poblet Martínez.

Until now these techniques had been used in radiological images. However, since the study of breast cancer using deep learning techniques, they are also being applied to other types of medical images. Such is their importance that several hospitals in the world have begun to digitize a large number of samples of all kinds for further processing.

The advancement of these technologies in the field of medicine causes a social revolution since it allows for diagnoses in a more efficient and faster way, thus being able to reduce the inconvenience that diagnosis times entails.

Being able to find and prioritize those cases that require primal diagnosis, could reduce the problems of some treatments due to not arriving on time; and in extreme cases could contrinute to save lives, granting a better quality of life.

A specific case has been chosen in collaboration with the Reina Sofía Hospital in Murcia and with Dr Enrique Poblet Martinez: "Detection of prostate cancer through biopsies". The idea is to be able to recognize cancerous areas in prostate biopsies in order to create a model capable of performing the screening process.

The images of biopsies are substantially different from the radiological images, as this is a field that is currently being investigated and in which only recently good results have begun to be achieved.

It is intended that this screening method be able to visually mark those areas with a high probability of containing cancer, with the aim of [in addition to prioritizing those that are more likely than those with less] providing hints on the location in which the cancer must be sought.

So far, the only studies that have been carried out were to verify whether Deep Learning techniques are capable of working in this field. Therefore, trying to develop a model that works for screening would be a great advance in this field.

A very important detail is the preference of a false positive rather to a false negative. Since the purpose of the model is to use it as a discriminator, it is desirable if a false negative is never obtained, causing the doctor to never have to look at it. If false negatives were allowed, a patient could end up with an erroneous diagnosis, while having false positives is less harmful since the physician can later correct the diagnosis if necessary, without any harm to the patient.

There is a very important stage which is commonly and largely omitted in mentioning, the preprocessing of the input images. In this case we have decided to cut the images, as has been done in other types of biopsies, due to the good results yielded.

Biopsies were scanned from a crystal at multiple magnifications and stored in a Tiff file for processing. Using the tools of the ImageMagick suite, the 20-fold magnification layer in Jpeg was extracted. These images have dimensions close to 5 GigaPíxeles (color RGB and 3 bytes per pixel), so they require some type of processing to be manageable. In this case we opted for cutting the image in tiles of 128x128 pixels, as this technique has already proven viable in the detection of breast cancer.

Even so, it is necessary to perform some type of preprocessing that may reduce the number of input variables, since the grater the number of variables, the more complicated the training task is. In this case we used convolution layers and slicing tiles. Later tiles were labelled by inference of the original image with the image.

After the tests were carried out, it can be seen that, with the proper studies a model with sufficient precision could be obtained to be used for screening in real scenarios.

For the training procedure, we had to generate a set of data for training and a set of validation. These sets had to be distributed as well, taking into account that the main goal of the networks is

to differenciate and fell apart those zones affected from cancer that those that are not, we estabished and 80% of the data for training and a 20% of the data for validation.

A doctor cannot be given a set of 0s and 1s, but a vision that is easier to understand and use, so a process that allows reconstructing the image of the biopsy is carried out, leaving those areas with a high probability of suffering from cancer duly labelled.

It is required that the number of false negatives be 0, so that no patients are left without a diagnosis. However, although false positive, are not desirable, doctors would still be able to correct the diagnosis when necessary.

In this work we have achieved a 78.1 % accuracy, which could be increased with necessary further studies. We have also managed to generate a methodology to follow up the study in this particular case.

Índice de figuras

Índice de tablas

1 Introducción

Resulta complicado no haber escuchado el concepto *Deep Learning* en los últimos años. Existen multitud de aplicaciones para este conjunto de técnicas de Inteligencia Artificial, sobre todo con el aumento de la capacidad de cómputo que permiten abordar problemas más y más complejos, siendo usado desde traductores inteligentes, reconocimiento de imágenes, hasta procesamiento del lenguaje.

Desde hace unos pocos años grandes empresas como Google, Intel, Amd, Nvidia, entre otras, están haciendo grandes inversiones en el campo de la inteligencia artificial.

Las redes neuronales son solo un tipo de Aprendizaje Máquina, el cual a pesar de que gran parte de las funciones que se usan actualmente fueron modeladas en los años 50-60, el mayor problema ha sido la falta de potencia de cómputo.

En el grupo de investigación GACOP (DITEC-Universidad de Murcia) llevan años trabajando en proyectos de HPC, pero con la importancia de los últimos años del *Deep Learning* se está trabajando en su aplicación, prueba de esto son los Trabajos de Fin de Grado de Antonio Matencio Escolar [1], José Antonio Bernabé Díaz [2], y posteriormente añadiéndose este mismo trabajo a la lista.

En el grupo han estado trabajando en procesadores Xeon, y las tarjetas Xeon Phi, pero en este caso usaremos tarjetas gráficas para el trabajo. Las redes neuronales realizan el mismo tipo de cálculo multiples veces, el ámbito donde las tarjetas gráficas destacan.

Este documento ha sido dividido en dos partes con el objetivo de resultar más sencillo de entender. En la primera parte veremos el funcionamiento de las redes neuronales y de los *frameworks*, mientras que la segunda parte se trabajará con un caso real usando los conocimientos aprendidos en la primera parte.

1.1 Motivación

El auge de las técnicas de *Deep Learning* en múltiples entornos crea un gran interés, y resulta interesante tener algunos conceptos y algunas experiencias con esta tecnología.

Debido a la gran precisión que es capaz de obtener en mucho ámbitos, se está usando: desde economía, videojuegos, hasta la medicina, siendo este último uno de los más interesante socialmente.

En el campo de la medicina se está usando para ser usado como discriminador, o como ayuda localizando el lugar a observar. El objetivo es acelerar y mejorar la calidad de los diagnósticos producidos por los médicos.

El caso que hemos tratado es el cancer de próstata, teniendo como objetivo obtener un discriminador capaz de detectar las zonas cancerígenas usando una biopsia de la misma, obtienendo en este trabajo un 78.2 % de *accuracy*. Para lograrlo hemos estado en colaboración con el Hospital Reina Sofía de Murcia y con el Dr. Enrique Poblet Martínez, quienes nos han otorgado muestras anonimizadas.

Parte 1

MXNET y Deep Learning

2 Fundamentos y estado del arte

2.1 Nacimiento de las Redes Neuronales

Aunque parezca algo totalmente novedoso, la gran mayoría de técnicas de Inteligencia Artificial (al menos su modelo teórico y matemático) aparecieron durante el siglo XX, pero la carencia de potencia de cómputo impidió su aplicación. En 1949, Donald Hebb escribió *"The Organization of the Behaviour"*, en el que se refleja la famosa "Ley de Hebb": ≪Si un elemento de procesado simple recibe una entrada de otro elemento de procesado y ambos están activos, el peso correspondiente a su conexión debería reforzarse≫ [3].

En 1959, Arthur Samuel estuvo estudiando que los ordenadores serían capaces de aprender sin necesidad de ser explicitamente programados. Para ello usó el juego de las damas, donde se le daba únicamente las reglas del juego, y un objetivo a cumplir [4], posteriormente estos estudios serían aplicables en otros campos.

A partir de estos estudios, en el mismo 1959, se realizaron dos modelos, "ADALINE" y "MADALINE" [5], ADAptative LINear Elements, y Many ADALINE, respectivamente. El funcionamiento de ADALINE era predecir el siguiente bit de la linea telefónica, y el objetivo de MADALINE era la reducción de eco en dichas lineas.

Sin embargo hasta ahora solo se habían tratado con redes de una sola capa, lo cual limita a problemas de clasificación separables linealmente, tal y como apunta Minsky y Papert en 1969 con *"Perceptrons: an introduction to computational geometry"* [6]. Y no fue hasta 1986 cuando se publicó el artículo *"Learning Representations By Back-Propagating Errors"* [7], en el cual se diseña un método de aprendizaje para redes neuronales multicapa, "back-propagation networks".

Con la publicación "Support-Vector Networks" [8] en 1995, en el cual se propone lo que se conoce como Máquinas de Soporte Vectorial, muestra que a grandes rasgos una Máquina de Soporte Vectorial busca un hiperplano que se separe de forma óptima de los puntos distribuidos en un espacio de múltiples dimensiones.

Aún con el gran avance del desarrollo de redes multicapa, el tiempo de aprendizaje necesario era demasiado elevado en la tecnología de la época, por lo que no fue hasta el siglo XXI con el avance de las nuevas arquitecturas (GP-GPU, MIC, procesadores multicore, ...) que se han podido emplear para grandes problemas reales, como la medicina, Big-Data, economía, y muchos más.

2.2 Redes Neuronales

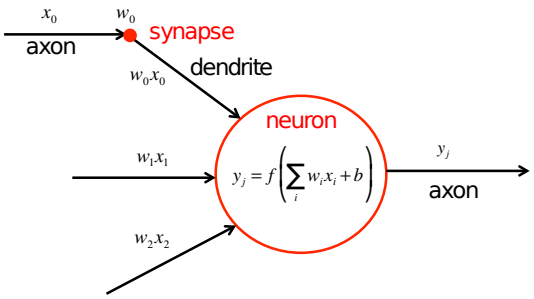


Figura 1: Esquema de una neurona artificial (perceptrón) en el contexto de las redes neuronales. Imagen extraida de [9].

Las redes neuronales son un tipo de aprendizaje máquina cuya base está en su unidad mínima de funcionamiento, la neurona artificial (perceptrón), la cual intenta imitar el comportamiento de las neuronas del ser humano.

Las investigaciones indicaron que existen, aproximadamente, 86 mil millones de neuronas en el cerebro humano. Estas se conectan entre sí usando dentritas (*dendrite*) y axones (*axon*), siendo las primeras las entradas (*input*) de la neurona, y el axón la salida (*output*). La neurona recibe una señal

por las dendritas, realiza un cómputo, y genera una señal de salida en el axón (Figura 1). La conexión entre una dendritas y un axón es denominado sinapsis (*synapse*), estimando que existen del orden de miles de billones en el cerebro humano [10].

En el cerebro humano, las entradas y la salida son un conjunto de señales eléctricas y químicas, sin embargo, en la neurona artificial, se representa como un número (normalmente en coma flotante). Esta entrada, dependiendo de dónde proceda, debe tener mayor o menor influencia, por lo que se multiplica por un número que se denomia peso (*weight*). La propia neurona tambien tiene una influencia global, por lo que se añade un *bias* (influencia o sesgo) a la salida.

Los perceptrones son capaces de representar funciones que relacionan las entradas con las salidas. Por ejemplo, si consideramos una neurona de dos entradas y una salida, los pesos $w_1 = w_2 = 1$, el *bias* a 0, y la función de activación $f(x) = x$, tendremos una neurona que es capaz de recibir 2 números y devolver la suma de estos.

Por sí mismos, de manera independiente, los perceptrones no son capaces de resolver problemas complejos, por lo que al igual que en el cerebro humano vamos a agruparlas para formar una red neuronal. Podemos conectar varios perceptrones entre sí uniendo la salida de uno con una o más entradas de otros perceptrones, organizándose en paralelo en grupos denominados capas (*Layer*), como se puede observar en la Figura 2.

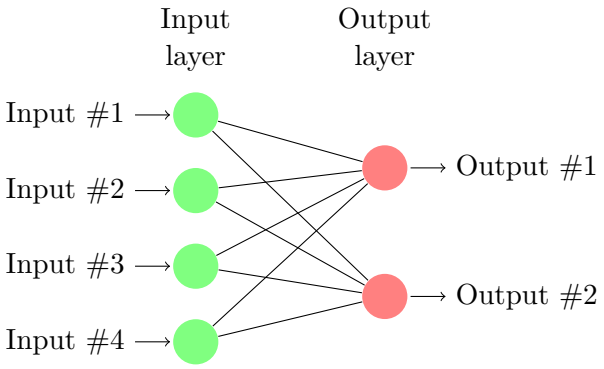


Figura 2: Esquema de una red neuronal de 2 capas.

2.3 Deep Learning

Cuando hablamos de 3 o más capas entra en juego el término Redes Neuronales Profundas (*Deep Neural Networks*), el cual hace referencia a que al menos existe una capa entre la capa de entrada y la de salida, las cuales reciben el nombre de capas ocultas (*hidden layers*). Estas capas ocultas permiten representar más información, extraer características, y abstraer los datos de entrada. La idea es ser capaz de modelar funciones más complejas de las que puede representar una red neuronal de 2 capas, pudiendo ampliar la cantidad de problemas que pueden abarcar.

Dentro del concepto de *Deep Learning* (Aprendizaje Profundo) existen múltiples técnicas, pero en este caso se hablará únicamente de las *Deep Neural Networks*.

2.3.1 MLP

Llamamos MLP (Redes Multicapa de Perceptrones o *MultiLayer Perceptron Network*) a las redes neuronales formadas únicamente por perceptrones, pero usando la idea del *Deep Learning*, por lo que disponemos de al menos una capa intermedia u oculta (Figura 3).

Las MLP son lo suficientemente potentes como para representar cualquier función que queramos enseñarle (con las suficientes capas ocultas) [11], incluyendo el reconocimiento de imágenes, texto, voz, entre otros. Sin embargo, para la gran mayoría de casos necesitariamos una gran cantidad de neuronas y capas, lo que requeriría de una excesiva capacidad de cómputo.

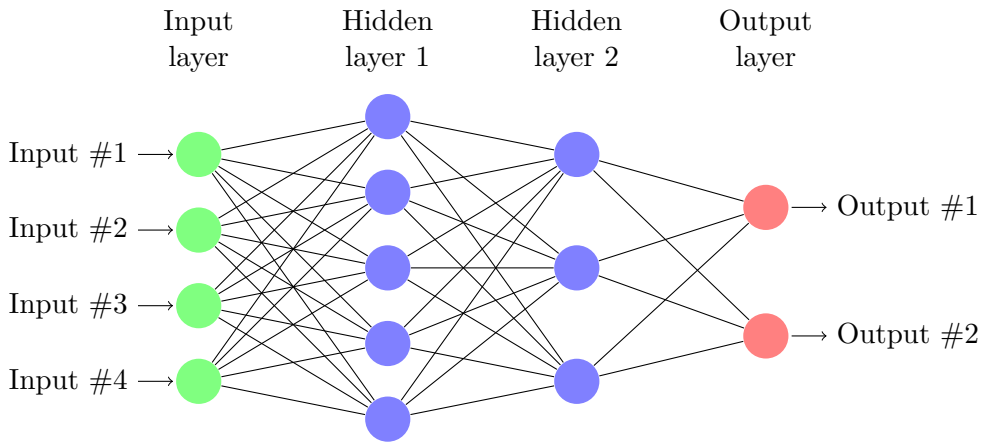


Figura 3: Esquema de una MLP de dos capas ocultas.

Ahora mismo estamos viéndolas como redes neuronales densas, es decir, cada capa tiene todas las conexiones posibles con la siguiente. Sin embargo, existen otro conjunto de redes que relajan esta regla y solo realizan ciertas conexiones: son las redes dispersas. Podemos ver estas redes dispersas como un intento de guiar aún más la red en el proceso de entrenamiento, ya que en muchos casos, las redes densas suelen tener una gran cantidad de enlaces con peso 0.

Una de las curiosidades de las redes neuronales, es la capacidad de unirse a otras. Es decir, si tenemos una red capaz de reconocer cifras y otra que relaciona las cifras, pueden unirse n copias de la primera red, y sus salidas estar conectadas a las entradas de la segunda red.

2.3.2 CNN

Según fueron avanzando las técnicas, se vio necesaria la creación de otros tipos de capas (además de los perceptrones), en definitiva las capas de perceptrones son capaces de representar distintas funciones, por lo que, se podría poner una capa con una función predefinida y no modificarla. Por ejemplo, realizar una operación lógica o matemática a las entradas. La ventaja del uso de estas capas es que no es necesario la modificación de sus pesos durante la fase de entrenamiento, y con el surgimiento del reconocimiento de imágenes empezaron a crease capas para filtrar y transformar las mismas.

La mayoría de CNN (*Convolutional Neural Networks*) suelen tener una conjunto de capas de convolución, con sus respectivas capas de *pooling*, seguidas de una red MLP para realizar el proceso de: clasificación, reducción, y transformación (Figura 4), aunque últimamente se empiezan a ver redes completamente convolucionales formando una estructura llamada *Encoder-Decoder*.

Una capa de convolución recibe como entrada un conjunto de datos estructurados en 2 dimensiones formando mapas de características, recibiendo el nombre de canales, el tamaño de estos canales reciben el nombre de dimensionalidad del canal. Por ejemplo, una imagen puede verse como 3 canales independientes representando cada uno la intensidad de uno de los colores rojo, verde, y azul [12]. La capa de convolución se encarga de leer la información de todos esos canales y generar una salida que mantiene la dimensionalidad original de cada canal, pero puede aumentar o disminuir la cantidad de canales.

Por otro lado, las capas de pooling trabajan cada canal de manera independiente. Su función es reducir la dimensionalidad de cada canal [12], recorrer el canal en tiles y devuelve un representante de dicho tile, ya sea mediante el máximo, el mínimo, una media, o cualquier otra función para obtener un número a partir de un conjunto mayor.

También existen otros tipos de capas que suelen ser aplicadas en convolución, aunque se suelen ver como metacapas. La más destacable es la de "flatter", la cual se encarga de recoger los datos en n dimensiones y reorganizarlos en un conjunto unidimensional, para que puedan ser usados en capas de otros tipos. Otras capas conocidas son las de deconvolución, cuyo objetivo es realizar la operación

inversa a la de convolución, y las de muestreo, que realiza la operación inversa al pooling.

Las redes neuronales de convolución intentan extraer características de las imágenes mediante el filtrado que realizan las capas de convolución, y además reducir la dimensionalidad del problema mediante las capas de pooling. Al realizar esta extracción de características pretendemos obtener conceptos más abstractos a partir de la imagen. Por ejemplo, en una red que reconoce caras, a partir de un número indeterminado de capas podríamos estar hablando de los conceptos: boca, ojos, nariz, entre otros.

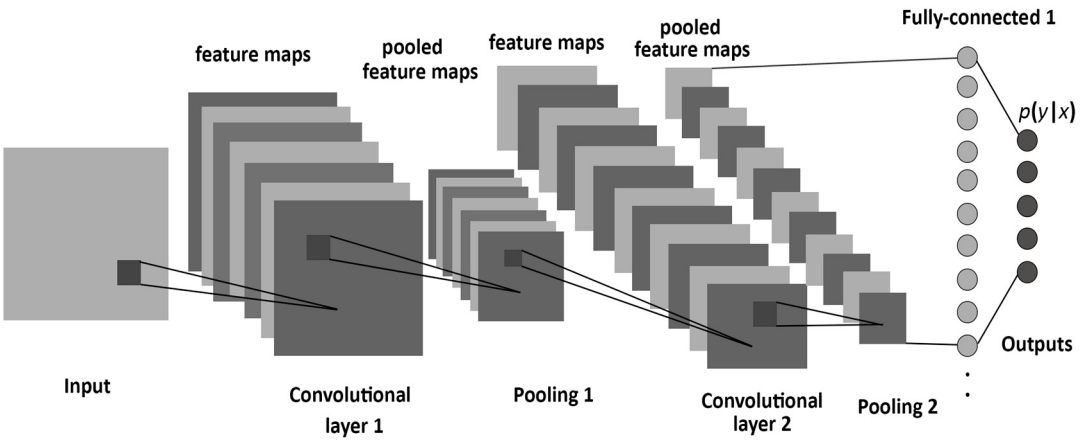


Figura 4: Esquema de una red neuronal de convolución
Imagen extraida de [13].

2.3.3 Otros tipos de redes

Existen otros tipos de redes neuronales, como por ejemplo las redes neuronales recurrentes (RNN), las cuales están formadas por copias de la misma red, y en los que en cada paso, se introduce una entrada, se generan dos salidas: una es la salida esperada de la red, y la otra es información para el siguiente paso de la RNN (Figura 5). Suelen usarse para reconocimiento del lenguaje, voz, y muchos otros tipos de problemas donde la entrada tiene un tamaño indeterminado y cada paso depende de los anteriores.

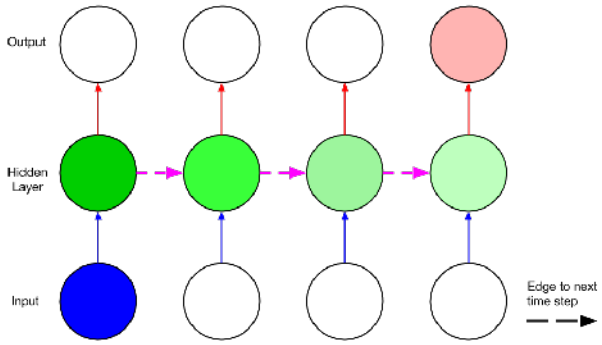


Figura 5: Esquema de una red neuronal recurrente
Imagen extraida de [14].

2.4 Training

Para que la red neuronal sea capaz de aprender algún patrón, esta deberá pasar por un proceso de entrenamiento. Existen multitud de tipos de entrenamientos para distintas áreas del aprendizaje máquina; en el caso de las redes neuronales, el más común es el aprendizaje supervisado, pero existen otros como el aprendizaje no supervisado.

2.4.1 Aprendizaje Supervisado

El algoritmo más usado y conocido es el SGD (*Stochastic Gradient Descendent* o Gradiente Descendiente Estocástico) el cual es una variante del GD (*Stochastic Gradient* o Gradiente Descendiente), pero aplicada por lotes. El algoritmo es un método iterativo para solucionar un problema de optimización. [15]

Este error se calcula mediante una función matemática que genera un valor para cada lote, las dos más conocidad son: el error cuadrático medio (MSE o *Mean Square Error*), y la exponencial normalizada (*SoftMax*).

El algoritmo SDG tiene 3 fases distintas por las que se iteran cada ejemplo: *FeedForward*, *BackPropagation*, y *Update*. Para que la red sea capaz de obtener un buen *accuracy* resulta necesario realizar varias iteraciones de los datos, a cada una de estas iteraciones se las conoce como épocas. Para reducir la probabilidad de caer en mínimos locales, los datos se presentan en lotes (*batches*) que son aleatorizados en cada época, y solo se realiza la fase *Update* después de cada lote.

La fase de *FeedForward* presenta cada uno de los datos de ejemplo a la red, infiriendo la salida a través de todas las capas para obtener la salida.

En la fase de *BackPropagation* se calcula el error producido entre la salida obtenida y el resultado esperado, iterando posteriormente el error desde la última capa hacia la primera, obteniendo una matriz δ, con los valores para corregir los pesos de cada neurona [15].

Por último la fase de *Update* actualiza los pesos teniendo en cuenta dos parametros *learning rate* (ratio de aprendizaje) y *momentum* (impulso), los cuales guían la influencia que va a tener la matriz δ sobre los pesos [15].

Es necesario tener cuidado con el número de épocas, ya que iterar demasiadas veces los mismos datos puede provocar un problema llamado *overfitting*, para el cual se han creado algunas técnicas para mitigarlo como por ejemplo una capa llamada *Dropout* [16].

También es importante el tamaño del lote, ya la fase de *Update* solo se realiza después de cada lote, por lo que si el tamaño es pequeño se realizan muchas *Updates*, y si es grandes se realiza pocas veces. Se ha realizado un estudio de como influye el tamaño de lote en el *accuracy* [17].

2.4.2 Aprendizaje No Supervisado

Al contrario que el aprendizaje supervisado, en el aprendizaje no supervisado los datos no disponen del valor esperado de salida, sino que la red debe encontrar el patrón por si mismo sin ayuda.

Este aprendizaje se basa en ir reforzando aquellas conexiones que son activadas en cada paso del entrenamiento independientemente del error de la red, pudiendose aplicar casi todos los conceptos del aprendizaje supervisado.

Para algunos problemas es capaz de obtener el mismo resultado en mucho menor tiempo que el aprendizaje supervisado.

2.5 Otros Conceptos

Existen múltiples formas de observar la precisión de un modelo concreto, siendo la más común el *accuracy* (porcentaje de acierto de la red), y la segunda más utilizada el *loss* (el error acumulado). Ambas son maneras similares de ver lo mismo: en el primer caso esperamos un número alto, cercano al 100 %, mientras que en el segundo esperamos un valor muy próximo a 0.

El comportamiento de la red puede verse alterador por multitud de variables, entre ellos podemos destacar: *learning rate*, *momentum*, *batch size*, *epochs*, además de muchas otras. Cada una altera el proceso de entrenamiento de distintas maneras, sin tener en cuenta el hecho de que además se pueden cambiar las funciones matemáticas que participan en la etapa de training.

Dependiendo de las funciones usadas en la etapa de training, también pueden influir unas variables que son conocidas como α y μ, que, al igual que todas las variables anteriores, influyen en cómo se realiza este proceso de entrenamiento.

El entrenamiento no se realiza desde 0, es importante un buen estado inicial. Originalmente se establecen todos los pesos de manera aleatoria, pero con el paso del tiempo se han desarrollado multiples técnicas, siendo la más conocida y estable la función *Xavius*, el cual inicializa los valores evitando que se vayan a los extremos de los intervalos.

3 Desarrollo del trabajo

3.1 Framework

Los frameworks otorgan un conjunto de políticas, librerias, herramientas, etc, para agilizar el desarrollo de software, por lo que actualmente no es necesario introducirse en este mundo creando las redes neuronales desde cero. Existen frameworks que facilitan la tarea de crear un modelo capaz de solucionar nuestro problema. La gran mayoría de frameworks existentes utilizan la misma base, o son muy similares, y suelen cambiar la API que ofrecen, la forma de instalarlo, o la integración en determinados sistemas más específicos. En la Figura 6 pueden verse los frameworks más conocidos en el ámbito de las redes neuronales, y en la Tabla 1 una pequeña comparativa.

Figura 6: Logos de algunos Frameworks para Redes Neuronales

Framework	Creator	License	OpenMP	Cuda	Multi-Node	Developed Language	Platform
TensorFlow	Google Brain	Apache	✓	✓	✓	Python, Cuda, C++	Windows, Linux, MacOs, Android
Theano	Montréal University	BSD	✓	✓	✓	Python	Cross-Platform
Caffe	Berkeley	BSD	✓	✓	✓	C++	Windows, Linux, MacOs
PyTorch	A. Paszke, S. Gross, ...	BSD	✓	✓	✓	C, Python, Cuda	Windows, Linux, MacOs
MXNet	Apache Foundation	Apache	✓	✓	✓	C++	Cross-Platform
Keras	François Chollet	Mit	Only with Theano Backend	✓	✓	Python, R	Windows, Linux, MacOs
CNTK	Microsoft	MIT	✓	✓	✓	C++	Windows, Linux, MacOs (Docker)

Tabla 1: Comparativa de algunos Frameworks de Redes Neuronales

3.2 HPC

Como ya se ha comentado, la fase de entrenamiento es la más costosa, en tiempo de ejecución, de las fases de desarrollo de una red neuronal, por lo que la gran mayoría de frameworks permiten el uso de tecnologías como Cuda, OpenMP, MPI, o similares, con el fin de usar los aceleradores o múltiples máquinas para reducir los tiempos de este proceso.

En cierta manera puede verse el entrenamiento de las redes como una búsqueda heurística de valores óptimos capaces de modelar la función matemática que representa el conjunto de datos con el que se desea entrenar, lo cual consume una gran cantidad de tiempo.

3.3 MXNet

En este caso se ha decidido por un framework de reciente creación MXNet, el cual fue acogido por el proyecto Apache Incubator, y que otorga soporte físico, personal, y económico a varios proyectos OpenSource.

Figura 7: Logo de MXNet

Se ha elegido MXNet como framework porque está recibiendo grandes apoyos de grandes empresas, ya que es un framework sencillo de instalar, 3 niveles de profundidad en la API, una muy buena documentación, entre otras cualidades. Permite una rápida adaptación desde otros frameworks ya que es capaz de trabajar con los archivos de configuración de estos.

Los frameworks exponen a los usuarios una interfaz para manejarlos, la mayoría de los frameworks de redes neuronales otorgan una interfaz de medio nivel de abstracción, otros ofrecen las de alto nivel, por ejemplo existe un framework llamado Keras que se basa en otros de medio nivel para otorgar una interfaz de alto nivel. MXNet otorga 3 niveles de abstracción, el más alto se conoce como Gluon, la de medio nivel permite hacer casi todo, y la de bajo nivel permite cambiar al detalle todas las opciones, incluso definiendo tus propias capas, y funciones.

MXNet está ofreciendo tan buen resultado que empresas como Intel están dandole soporte en las plataformas existentes e incluso en las nuevas, por ejemplo Intel OpenVino es capaz de desplegar modelos de Caffe, TensorFlow, y MXNet en CPU, GPU, VPU, y FPGA [18]. No siendo esta la única, otro ejemplo es Amazon Web Services, que otorga soporte para el entrenamiento y el despliegue de modelos a traves de la API Gluon [19].

MXNet intenta maximizar la eficiencia y la productividad, en su núcleo contiene un planificador dinámico para paralelizar las operaciones imperativas y las simbólicas. Trata de ser lo más portable posible y liviano, escalando eficientemente entre múltiples GPUs y múltiples máquinas.

3.3.1 Creación de las Redes

Debido a la API de 3 profundidades se puede desarrollar la red de multiples maneras: la primera usando archivos de configuración (prototxt, json, entre otros), y la segunda programando la red usando algunas de las profundidades de la API en uno de los lenguajes aceptados, siendo este último el que se usará en este trabajo.

Los datos a introducir en la red deben estar indexados siguiendo cierto formato, MXNet nos permite usar varios: *image lists*, *csv*, *recordIO*, entre otros, siendo este último muy interesante para mejorar la lectura desde HDD, sin embargo debido al coste del preprocesamiento solo debe ser usado en grandes bases de datos ya preparadas para el entrenamiento.

3.3.2 Instalación

Se va a trabajar usando GPUs, yendo a la página oficial de MXNet (https://mxnet.apache.org/install/index.html) y seleccionando la opción elegida, en este caso Linux, Python, GPU, y Pip. El proceso de instalación se detalla en el Anexo I, partiendo de un sistema completamente funcional con todos los requisitos.

Al empezar este trabajo, el framework estaba en su versión 0.12, sin embargo según avanzaba el tiempo se vio la necesidad de usar las nuevas versiones por los arreglos y mejoras que estas fueron trayendo, sin embargo actualizar el framework es igual de sencillo que instalarlo.

Una vez instalado es necesario comprobar que todo funciona correctamente, para ello se comprobara mediante 2 modelos muy bien conocidos: MNIST, y CIFAR-10.

4 Evaluación y discusión

Existen dos bases de datos muy utilizadas para estudiar los cambios que pueden provocar: ciertas modificaciones a los frameworks [20], nuevos tipos de capas [16], cambios de precisión [21], influencia de ciertos parámetros [17], y muchos otros. El objetivo de estas bases de datos es otorgar un conjunto de datos reales que puedan ser usados fácilmente por cualquier persona.

4.1 MNIST

El MNIST[1] es una base de datos (subconjunto de una más grande) de dígitos escritos a mano que dispone de 60 000 imágenes para entrenamiento, y 10 000 imágenes para validación.

Para su desarrollo, cientos de personas escribieron en una página dentro de rectángulos los números que se le fueron pidiendo. Posteriormente los números digitalizados en cuadrados de 20x20 píxeles que fueron enmarcados en 28x28 píxeles centrando el número. Además solo se almacenan en escala de grises, y limitandolo a 256 tonos (1 byte), y empaquetado en un único fichero con toda la información necesaria para reconstruir las imágenes (Figura 11).

Figura 8: Un pequeño ejemplo del MNIST
Imagen extraida de [22]

4.1.1 Red Neuronal

El ejemplo de la red (Figura 9) es una MLP que debe ser capaz de otorgar un 96 % de *accuracy*, por lo que tras el entrenamiento se deberían obtener valores muy similares. Esta prueba fue realizada en la versión de MXNET 0.10, por lo que pueden haber pequeñas diferencias con las versiones actuales.

El framework nos ofrece una configuración recomendada para obtener el *accuracy* esperado. La entrada tiene 784 neuronas, que es exactamente 28x28, es decir cada neurona tiene la información de un píxel de la imagen en escala de grises de 256 tonos. La segunda y tercera capa tienen una función "relu" con 128 y 64 neuronas respectivamente, siendo seguidas por la capa de salida, con la función *softmax*, capaz de clasificar en 10 categorias, es decir los 10 dígitos. Por último tenemos una capa extra para medir la precisión de la red, y como soporte al proceso de *BackPropagation* mediante el error cuadrático medio.

Como parámetros adicionales se usaron un *learning rate* de 0.05, que disminuye cada 10 épocas, un total de 20 épocas, y 64 imágenes por *batch*.

[1] Más información: http://yann.lecun.com/exdb/mnist/

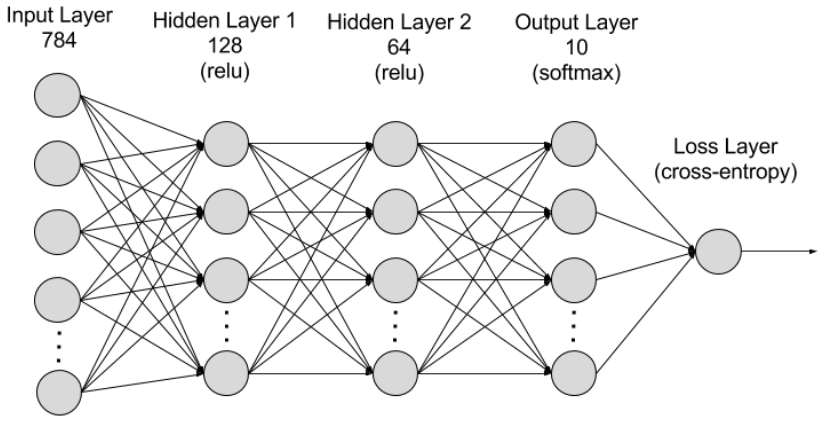


Figura 9: Red de ejemplo para MNIST de MXNET
Imagen extraida de https://mxnet.incubator.apache.org/tutorials/python/mnist.html

4.1.2 Prueba de ejecución

Una vez todo entendido, es necesario comprobar si se obtiene el resultado esperado. La documentación indica que el mínimo es de 96 % de accuracy, sin embargo estos resultados son de la versión 0.10 de MXNET, por lo que con la mejora del framework hasta la actual 1.2 esperamos el mismo resultado o incluso mejor.

```
INFO:root:Epoch[19] Batch [100] Speed: 33298.82 samples/sec accuracy=1.000000
INFO:root:Epoch[19] Batch [200] Speed: 33419.54 samples/sec accuracy=0.999375
INFO:root:Epoch[19] Batch [300] Speed: 33423.95 samples/sec accuracy=0.999687
INFO:root:Epoch[19] Batch [400] Speed: 33405.07 samples/sec accuracy=0.999687
INFO:root:Epoch[19] Batch [500] Speed: 33387.70 samples/sec accuracy=0.999687
INFO:root:Epoch[19] Batch [600] Speed: 33431.53 samples/sec accuracy=0.999531
INFO:root:Epoch[19] Batch [700] Speed: 33379.98 samples/sec accuracy=0.999687
INFO:root:Epoch[19] Batch [800] Speed: 33417.50 samples/sec accuracy=1.000000
INFO:root:Epoch[19] Batch [900] Speed: 33444.02 samples/sec accuracy=0.999844
INFO:root:Epoch[19] Train-accuracy=0.999155
INFO:root:Epoch[19] Time cost=1.800
INFO:root:Epoch[19] Validation-accuracy=0.981986
```

Al ver la salida de la última época, en el Anexo II pueden observarse una tabla y gráfica con el accuracy en cada época. Se obtiene una precisión en la validación del 98 %, lo cual demuestra que además de que el framework funciona, se han ido realizando mejores que aumentan la precisión.

4.2 CIFAR

CIFAR[2] (Canadian Institute For Advanced Research en honor por el grupo que lo recopiló) es otra base de datos muy usada a modo comparativo, en este caso trata de clasificar un conjunto de imágenes en n categorias. Existen distintos conjuntos, siendo los más conocidos CIFAR-10 y CIFAR-100, el número indica el número de categorías a las que pertenecen las imágenes a clasificar.

En este caso se usará CIFAR-10, el cual contiene 6.000 imágenes de cada clase, siendo estas de 32x32 píxeles a color (Figura 10). Normalmente vienen empaquetadas en una base de datos, o en una jerarquía de carpetas (1 carpeta por clase). Sin embargo al derivar de CIFAR-100 estas clasificaciones

[2]Más información: https://www.cs.toronto.edu/~kriz/cifar.html

pueden cambiarse por otras, e incluso en otros casos alterar la dimensionalidad de dichas imágenes con el fin de adaptarse a casi cualquier red que queramos entrenar.

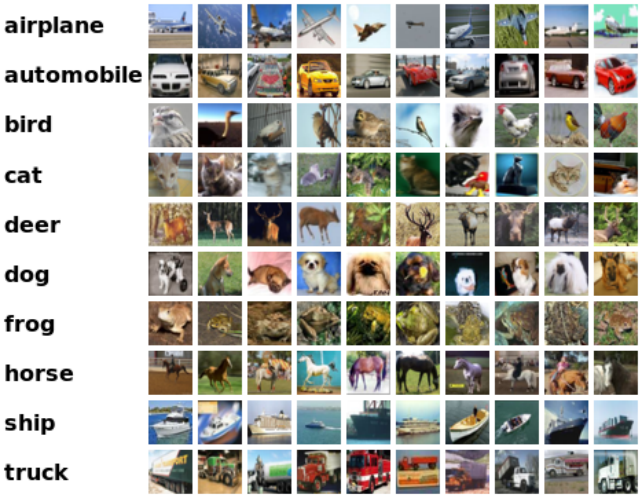


Figura 10: Un pequeño ejemplo del CIFAR-10
Imagen extraida de https://www.cs.toronto.edu/~kriz/cifar.html

4.2.1 Red Neuronal

La red usada para este problema es la ResNet-110, en anteriores versiones de MXNet el ejemplo estaba formado por ResNet-101. La red ResNet es una red modular creada por bloques más sencillos [23] (Figura 11). El número que acompaña al nombre de la red indica el número de capas que tiene dicha red, además existen un conjunto de variables de los bloques forman, que alteran la configuración de los mismos. A la red formada se le añaden 2 capas, una al principio para manejar la entrada, y otra en la salida para realizar la clasificación.

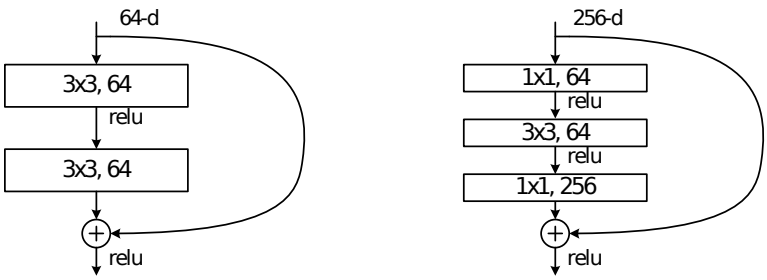


Figura 11: Bloque de construcción de una ResNet
Imagen extraida de [23]

Se puede observar que la red está formada por capas de convolución con funciones *relu*, que acaban en una capa *relu* que utiliza información extraida de la convolución con información anterior a la convolución.

El framework nos ofrece una configuración para obtener un buen *accuracy*: red ResNet-110 usando imágenes de 28x28 en lugar de 32x32, en *batches* de 128 imágenes, con un total de 300 épocas, un *learning rate* de 0.05, que disminuye en las épocas 200 y 250. El conjunto de entrenamiento es de 50 000 imágenes, y 10 000 para validación.

4.2.2 Prueba de ejecución

Las ejecuciones de los ejemplos de MXNet esperan un resultado cercanos al 85-90 % en su versión 0.10 usando ResNet-101, por lo que la nueva configuración debería dar un mejor *accuracy*, al igual que en

MNIST.

```
INFO:root:Epoch[299] Batch [20] Speed: 1391.08 samples/sec accuracy=0.994420
INFO:root:Epoch[299] Batch [40] Speed: 1347.21 samples/sec accuracy=0.997656
INFO:root:Epoch[299] Batch [60] Speed: 1351.73 samples/sec accuracy=0.996094
INFO:root:Epoch[299] Batch [80] Speed: 1371.94 samples/sec accuracy=0.996094
INFO:root:Epoch[299] Batch [100]    Speed: 1369.48 samples/sec   accuracy=0.995313
INFO:root:Epoch[299] Batch [120]    Speed: 1348.64 samples/sec   accuracy=0.994922
INFO:root:Epoch[299] Batch [140]    Speed: 1352.53 samples/sec   accuracy=0.991797
INFO:root:Epoch[299] Batch [160]    Speed: 1376.67 samples/sec   accuracy=0.994922
INFO:root:Epoch[299] Batch [180]    Speed: 1380.04 samples/sec   accuracy=0.995703
INFO:root:Epoch[299] Batch [200]    Speed: 1346.67 samples/sec   accuracy=0.994141
INFO:root:Epoch[299] Batch [220]    Speed: 1346.05 samples/sec   accuracy=0.998047
INFO:root:Epoch[299] Batch [240]    Speed: 1356.32 samples/sec   accuracy=0.995703
INFO:root:Epoch[299] Batch [260]    Speed: 1345.42 samples/sec   accuracy=0.991797
INFO:root:Epoch[299] Batch [280]    Speed: 1374.77 samples/sec   accuracy=0.993750
INFO:root:Epoch[299] Batch [300]    Speed: 1353.37 samples/sec   accuracy=0.996484
INFO:root:Epoch[299] Batch [320]    Speed: 1361.51 samples/sec   accuracy=0.995703
INFO:root:Epoch[299] Batch [340]    Speed: 1343.11 samples/sec   accuracy=0.996875
INFO:root:Epoch[299] Batch [360]    Speed: 1363.11 samples/sec   accuracy=0.997656
INFO:root:Epoch[299] Batch [380]    Speed: 1359.42 samples/sec   accuracy=0.996484
INFO:root:Epoch[299] Train-accuracy=0.995313
INFO:root:Epoch[299] Time cost=36.790
INFO:root:Epoch[299] Validation-accuracy=0.931791
```

En la salida del entrenamiento se obtiene un *accuracy* del 93 %, demostrando que el framework funciona correctamente, se puede observar una tabla y un gráfico del accuracy por época en el Anexo III.

Parte 2

Aplicación en un caso médico real

5 Fundamentos y estado del arte

Se ha visto que en los últimos años las tecnologías basadas en deep learning están teniendo mucho peso en grandes áreas, provocando una gran revuelta[3]. Entre dichas áreas hay que destacar una muy importante, la medicina.

5.1 Deep Learning en medicina

Hasta hace muy poco, la aplicación de técnicas de Deep Learning se limitaba a imágenes radiológicas [24], sin embargo, en los últimos años (2016-2018) se están empezando a aplicar a otros campos de la medicina, desde "Detección de desprendimiento de retina" [25], hasta el caso más conocido "Detección del cancer de mama" [26].

Debido a esto, desde hace unos años varios hospitales empezaron a digitalizar todo tipo de muestras con el fin de poder ser usadas con varios tipos de técnicas de Machine Learning, ya que el mayor problema en este campo son los datos, que suelen escasear. Por ejemplo existen 3 bases de datos de imágenes de retinas (E-Ophtha, ROC, DIARETDB1) para el estudio de detección de microaneurisma a traves de la retina [27].

Sin embargo la detección de zonas extrañas no es el único campo en el que se está trabajando, también se está usando con cadenas de ADN y ARN [28].

El avance de estas tecnologías en el campo de la medicina provoca una revolución social ya que permite diagnosticar de una manera más eficaz y rápida. Pudiendo reducir los problemas que conlleva los tiempos de diagnóstico.

Pudiendo encontrar y priorizar aquellos casos que lo requieran podría reducir los problemas de algunos tratamientos por no llegar a tiempo, y en casos extremos ayudar a salvar vidas, otorgando una mejor calidad de vida.

5.2 Caso elegido

Se ha elegido un caso concreto, en colaboración con el Hospital Reina Sofía de Murcia y con el Dr. Enrique Poblet Martinez, "Detección de cancer de próstata mediante biopsias". La idea es reconocer zonas cancerígenas en las biopsias de próstata con el fin de crear un modelo capaz de realizar el proceso de *screening*.

Las imágenes de biopsias son sustancialmente distintas a las imágenes radiológicas, siendo un campo que se está investigando y en el que se está empezando a obtener un buen resultado.

Se desea que este método de *screening* sea capaz de marcar de manera visual aquellas zonas con alta probabilidad de contener cancer, con el objetivo de, que además de priorizar aquellas que tienen más probabilidad de las que tienen menos, proporcionar una ayuda del lugar en el que se ha de buscar dicho cancer.

Hasta ahora solo se han realizado estudios para comprobar si las técnicas de *Deep Learning* son capaces de funcionar en este campo, por lo que intentar desarrollar un modelo que funcione para *screening* sería un gran avance en dicho campo.

Una nota muy importante es la preferencia del falso positivo al falso negativo, al querer usarlo como discriminador, sería deseable que nunca se obtuviese un falso negativo provocando que el médico nunca tuviese que mirarlas. Si permitiesemos el falso negativo, un paciente podría acabar un diagnóstico erróneo, mientras que tener falsos positivos es menos perjudicial ya que posteriormente el médico puede corregir el diagnóstico en caso de ser necesario, sin problemas para el paciente.

[3] Existen multitud de competiciones, muchas de ellas con datos médicos en https://www.kaggle.com/.

6 Desarrollo del trabajo

6.1 Imágenes

El primer paso del trabajo es observar como son las biopsias, estando contenidas en un cristal, que se observa a microscopio para generar el diagnóstico, siendo necesario la digitalización dichas biopsias para trabajar con ellas usando unos escáneres especificamente creados para ello.

6.1.1 Digitalización

Antes de ser escaneadas, se han anonimizado las muestras para preservar la privacidad de los pacientes (Figura 12), en concreto las muestras llevan el nombre en la primera imagen del cristal, por lo que se ha difuminado dicha área.

Una vez digitalizados, el escaner nos ofrece distintos formatos de salida para la imagen resultado: Biff, Tiff (o Tif), y Jpeg 2000. Biff es un formato bastante complicado de manejar, siendo descartado por otros formatos más viables. Jpeg 2000 es una buena opción, pero a la hora de procesarlo, el algoritmo de decodificación consume demasiada memoria y tiempo. Tiff ofrece un balance muy bueno entre tamaño y calidad, por lo que después de probar con el Jpeg 2000 se convirtió en la opción más adecuada.

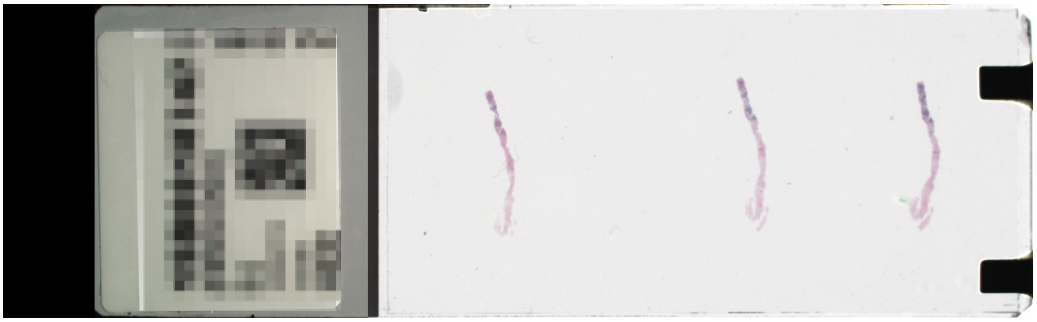


Figura 12: Un cristal de una biopsia de próstata

El escaner realiza varias digitalización de las imágenes a distintos niveles de ampliación: 1x, 2x, 5x, 10x, 20x, y 40x, por lo que en caso de necesitarlo podríamos cambiar la escala de trabajo. Además el escaner incluye una lente fija de 10x, siendo todos los números anteriores multiplicados por 10.

6.1.2 Dimensiones de las Imágenes

Según se usan aumentos mayores, el tamaño de la imagen aumenta, tanto en peso como en dimensiones, pudiendo resultar inviable entrenar una red neuronal usando las imágenes completas, estando comprendidas estas en dimensiones cercanas a los 5-10 Gigapixels, aunque depende mucho de cada biopsia.

El problema del peso de la imagen es relativamente fácil de solucionar, ya que la imagen está codificada en jpeg y la mayor parte de la misma es blanca, por lo que a 20x puede resultar en tamaños de 100-200 MiB. Sin embargo, cuando dichas imágenes se cargan en memoria para ser procesadas, llegan a tamaños de 3-4 GiB por imagen, lo cual complica su uso.

6.2 Tratamiento de las imágenes

Conociendo todos los problemas, y cómo son las imágenes, se va a utilizar el formato Jpeg extraido del Tiff. Sin embargo, sigue siendo muy complicado el entrenamiento de la red utilizando imágenes de dicho tamaño, por lo que se ha utilizado la suite de ImageMagick, y en algunos casos con ayuda de GNU Parallel [29].

6.2.1 Formato y Corte

En primer lugar se extraerá la ampliación que se va a usar (en este caso 20x) del tiff, posteriormente seleccionando la capa de la biopsia a procesar (Figura 13). Para ello se usará un conjunto de utilidades que vienen agrupados por ImageMagick, el cual es capaz de trabajar con imágenes de tan grandes dimensiones.

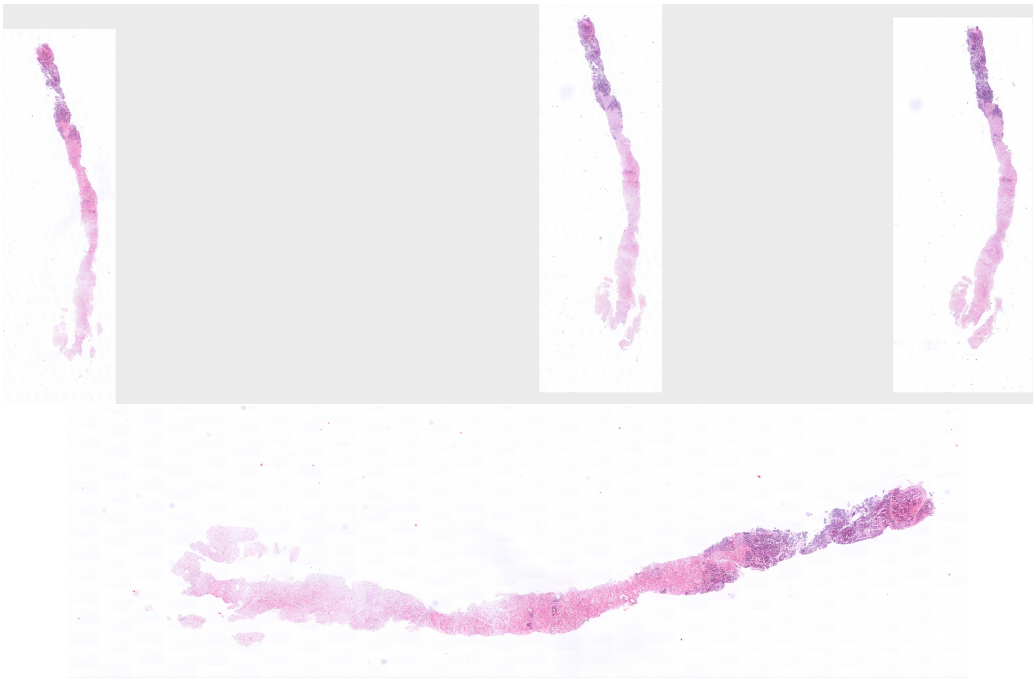


Figura 13: Una biopsia, y la capa de la biopsia que nos interesa

A la hora de entrenar una red neuronal, si la entrada es demasiado grande produce varios problemas: el primero es la cantidad de memoria que consume la red para entrenarla, el segundo es el gran conjunto de variables puede provocar que la red no sea capaz de aprender el patron, siendo necesario reducir esta entrada otorgando los valores significativos. Sin embargo, la necesidad de reducir esta entrada depende del problema concreto a resolver.

El problema del tamaño de la entrada puede ser mitigado mediante distintas técnicas, desde un gran conjunto de capas de convolución y *pooling* para reducir el tamaño rapidamente, reducir la propia imagen con otras técnicas antes de pasar por la red [25], o la técnica que se va a aplicar, que ha sido empleada en dos casos: el primero es el más conocido: “Detección del cancer de mama” [26], el segundo es un intento de ver si es viable el uso de estas técnicas en dos tipos de cancer distintos: el cancer de próstata y un tipo concreto de cancer de mama [30].

Una vez obtenida la biopsia deseada en jpeg, se envia de vuelta la imagen a los patólogos para que se realize un marcado con un programa de edición de imágenes similar a Gimp o Photoshop (Figura 14).

La técnica usada es el trozeado de las imágenes en *tiles*, la cual tiene sus ventajas y desventajas dependiendo de como se aplique, y se quiera obtener. La ventaja más evidente es que la dimensión de la imagen pasa a ser indiferente. Por otro lado, la desventaja más problemática es la perdida de la visión global del problema, al cortar la imagen en trozos puede ocurrir que la red no sea capaz de encontrar las zonas cancerígenas, por lo que hay que tener especial cuidado en el tamaño de dichos *tiles*.

Para este problema se ha usado el tamaño de 128x128 para los *tiles*, por ser un tamaño no excesivamente pequeño. Es necesario realizar un estudio de la dimensión de los *tiles*, pero en este trabajo se usará el tamaño sugerido 128x128 [26].

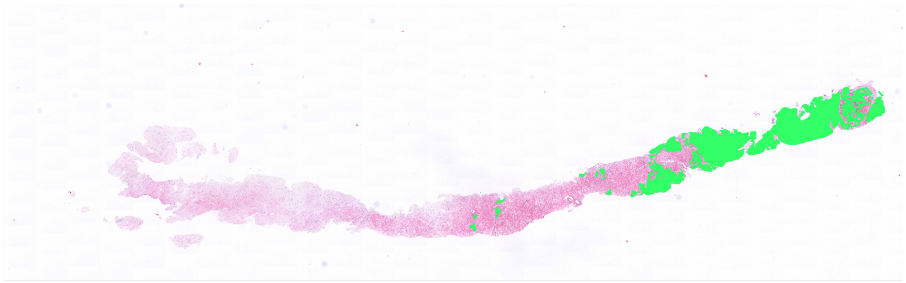
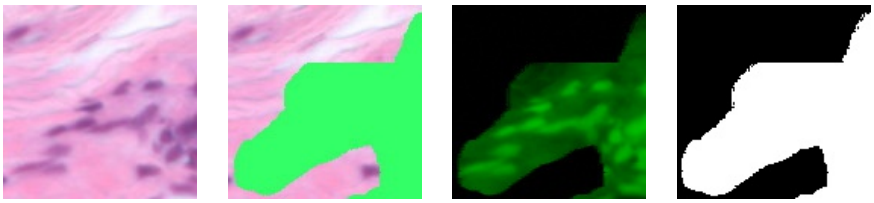


Figura 14: Una biopsia marcada

6.2.2 Etiquetado de las zonas cancerígenas

Una vez troceado la biopsia, y la biopsia marcada, se aplicará OpenCV con su *binding* en Python para detectar el color de la zona marcada en cada *tile*, si este lo tiene. Para ello, se carga el *tile* a comparar y su homónimo marcado, realizando la diferencia, y posteriormente un *threshold* con el fin de eliminar posible ruido (Figura 15). Por último se obtiene la media del color de todos los píxeles dentro de la zona blanca, pero cogiendo los píxeles del *tile* marcado.



Original Marcada Diferencia Filtrado

Figura 15: Comparación de un *tile*

Con el objetivo de poder marcar más de un tipo de cáncer, la etiqueta del *tile* se establece en función del color, pero para este trabajo solo se etiqueta como cáncer o no cáncer.

Como nota adicional, todos los *tiles* que incluyan al menos una pequeña parte de cáncer, aproximadamente 5 %, son marcados como cáncer. Se ha elegido este tipo de marcado con el objetivo de entrenar la red incitando el Falso Positivo ante el Falso Negativo.

6.3 Red Utilizada

Para empezar se va a partir de la red neuronal usada en el estudio sobre el uso del Aprendizaje Profundo para el diagnóstico histopatológico [30], pero realizando modificaciones según se han ido observando problemas.

6.3.1 Estructura de la red

En la Figura 16 se puede ver la red usada en este caso, en el cual se observan 2 capa de convolución, 2 capas de pooling, y 3 capas formando una MLP.

La red contiene 2 neuronas para indicar la salida de la red, en este caso la primera indica “sin cáncer” y la segunda “con cáncer” que podría usarse una única salida, pero posteriormente se pretende poder representar más de 1 tipo de cáncer.

Existen unas metacapas que no se pueden observar en la imagen de la red: la primera se encuentra al principio de la red y tiene la función de dividir todos los valores de la entrada por 255 con el objetivo de que todos los valores se encuentren entre 0 y 1, la segunda es una capa de aplanamiento que se encuentra antes de la MLP.

Al realizar un problema de clasificación, las salidas se encuentras con valores entre 0 y 1, por lo que al llevar la entrada a dicho rango de valores simplifica el problema del aprendizaje.

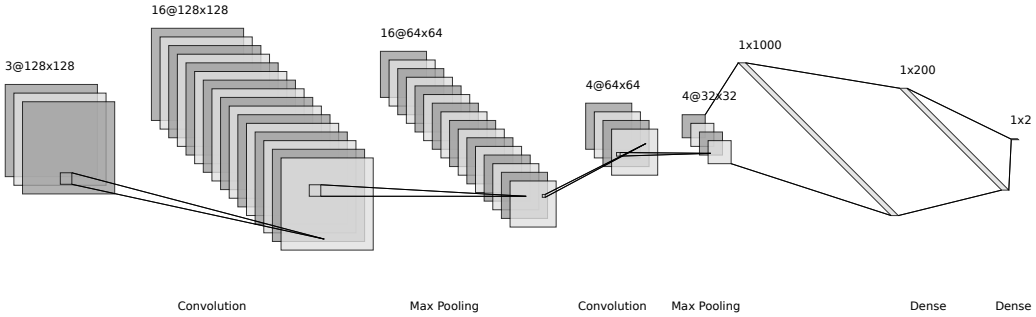


Figura 16: Red neuronal usada
Generada usando http://alexlenail.me/NN-SVG/LeNet.html

Las capas de pooling utilizan la funciona máximo para realizar la reducción de la dimensionalidad, y las capas de la MPL tienen la función sigmoid. Al final el cálculo del error para el *BackPropagation* se realiza mediante una función *Softmax*.

6.3.2 Parámetros

Se han utilizado los siguientes parámetros para obtener un 78.2 % de *accuracy*: *learning rate* de 0.001, *momentum* de 0.8, y un tamaño de lote de 32 imágenes.

6.4 Set de Training y Validación

Actualmente se disponen de 18 imágenes preparadas, es decir, imágenes totalmente marcadas y clasificadas, adicionalmente se tienen 3 imágenes guardadas para una comprobación visual de los resultados.

Durante el entrenamiento se usarán 17 imágenes, y la validación se hará con la restante de las imágenes preparadas, lo cual sumado a las 3 imágenes de comprobación visual hace aproximadamente un 20 % de validación y un 80 % de entrenamiento.

6.4.1 Separación de las Imágenes

Al principio se usaban todos los *tiles* que teníamos para entrenar la red, lo cual además de no obtener un buen resultado requería de una gran cantidad de épocas en empezar a aprender parte del patrón. Provocando que se requiriera de un mejor reparto del set de entrenamiento, siendo la primera elección eliminar aquellos *tiles* que son completamente blancos.

Reparto (No cáncer - Cáncer)	30-70	50-50	70-30	All
Con Blancos	46.9 %	57.5 %	49.1 %	32.7 %
Casi sin Blancos	59.2 %	78.2 %	57.7 %	49.0 %

Tabla 2: Comparativa de reparto usando 100 épocas

Al realizar la comparativa se puede observar que la cantidad de imágenes que se le ofrecen en la etapa de trainning tiene una influencia considerable en el tiempo que tarda en converger, ya que casi todos eran capaces de llegar pero aumentando de manera importante la cantidad de épocas (Tabla 2).

Para eliminar los *tiles* blancos se realizó otro programa mediante Python y OpenCV que recorre todos los píxeles para saber si estos son blancos, sin embargo queriamos mantener una pequeña cantidad de estos.

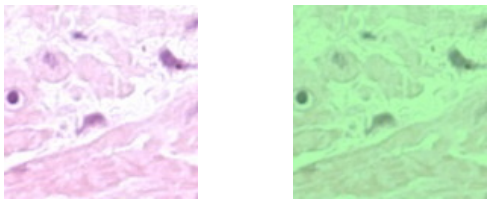
La idea al elegir los porcentajes del reparto era intentar guiar a la red sobre que es lo que debe buscar. Queriendo que sea capaz de distinguir entre cáncer y no cáncer, pero siendo capaz reconocer aquellos *tiles* que sean blancos sin prestarles especial atención, por lo que el programa deja aproximadamente un 2 % de imágenes blancas (o casi blancas) en la salida.

6.5 Postprocesado

La idea final es generar un proceso que pueda servir de *screening*, provocando que la salida tenga que ser sencilla de usar por un médico, por lo que no se puede devolcer una matriz de 0s y 1s como resultado.

6.5.1 Alpha Blending

Es necesario marcar aquellas zonas que son prometedoras de tener cáncer, por lo que se realizará un proceso *alpha blending* con un color y los *tiles* que lo requieran, por lo que se usará otro programa, desarrollado en Python y OpenCV, que será capaz de interpretar la salida de la red neuronal y marcar correctamente los *tiles* que lo requieran (Figura 17).

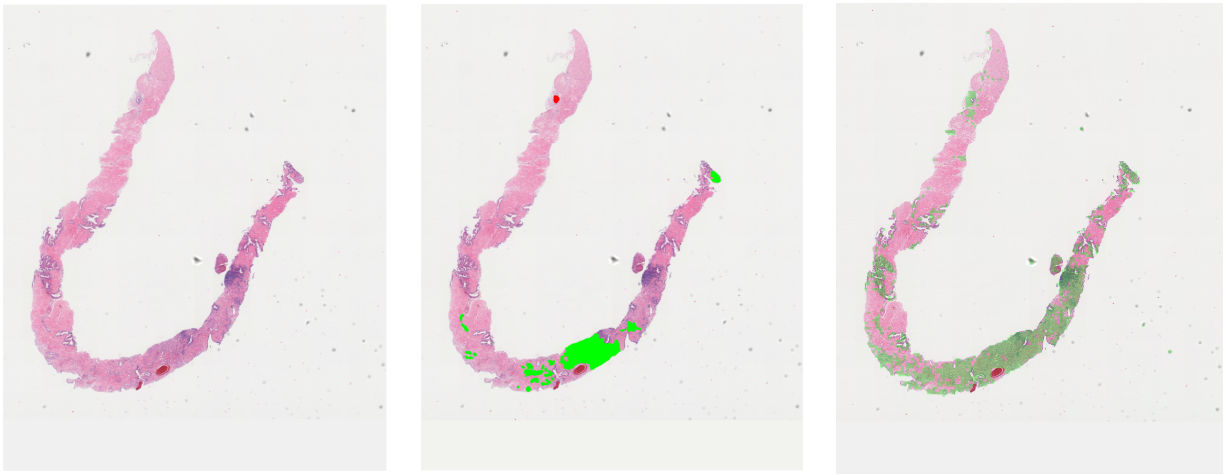


Antes del Blending Después del Blending

Figura 17: Blending de un *tile* cancerígeno en verde

6.5.2 Reconstrucción de la imagen

Sabiendo el tamaño de la imagen final es sencillo realizar una reconstrucción mediante el programa *montage*, el cual forma parte de la suite de ImageMagic (Figura 18).



Original Marcada (Médicos) Marcada (Red)

Figura 18: Reconstrucción de la imagen detectada por la red

7 Evaluación y discusión

7.1 Tiempos

La máquina usada está equipada con una CPU Intel(R) Xeon(R) CPU E5-2603 V3 @1.6GHz, 64GB de Memoria DDR4 a 2133MHz, Nvidia GTX 1080 Pascal, y SSD Samsung Evo 500 GB. En la Tabla 3 se puede observar los tiempos de cada fase del trabajo, los cuales se redujeron bastante desde el uso de un SSD en lugar de un HDD tradicional.

Tarea	Tiempo
Preprocesamiento	
Tiff → Jpeg	45s
Extraer Biopsia	12s
Etiquetado	12m
Generación de listas (Base de Datos)	37s
Limpieza de Blancos	3h 45m
Entrenamiento de la Red	
Training 50 épocas	47m
Training 100 épocas	1h 16m
Predicción	
Tiff → Jpeg	47s
Extraer Biopsia	13s
Cortado	1m
Predicción	2m
Alpha-Blend	17m
Reconstrucción	27s

Tabla 3: Tiempos de ejecución

Podemos observar que la estapa de preprocesamiento tiene una duración cercana a las 4h, generando una base de datos que puede usarse para el entrenamiento de la red. Dicho entrenamiento llega a tiempos de entre 1 a 2 horas, obteniendo un 78 % de *accuracy*.

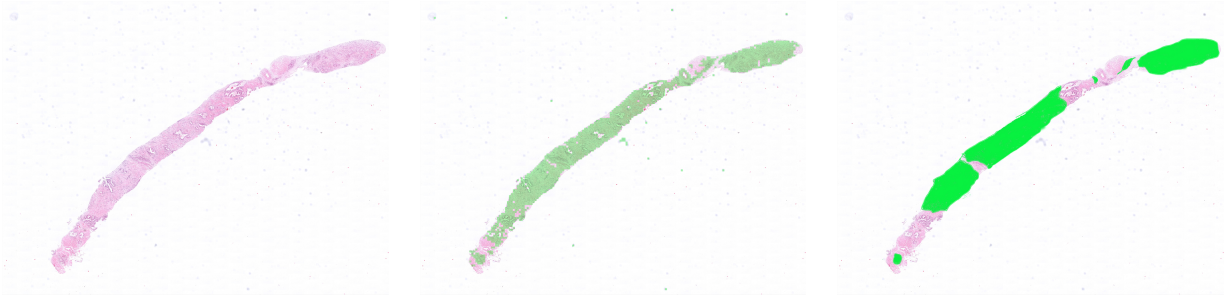
En la ejecución de un caso simulando una predicción real, el preprocesamiento lleva tiempos cercanos a 2 minutos. La predicción de la red lleva otros 2 minutos, pudiendo hablar casi de tiempo real. Por último, la reconstrucción de la imagen con el Alpha Blending puede conlleva casi 20 minutos, pero esta fase es añadida por exigencias médicas.

7.2 Resultados

A continuación se puede ver la comparativa de las 4 imágenes de test, con sus respectivas tablas de confusión, usando la configuración de la red que otorga el 78.2 %. Los casos han sido etiquetados con un número único, en este caso, para facilitar la comunicación con el Hospital Reina Sofía se han mantenido los números que utiliza el escáner.

		True diagnosis		
		Negative	Positive	Total
Screening test	Negative	27343	198	27541
	Positive	299	1264	1563
	Total	27642	1462	29104

Tabla 4: Matriz de confusión del caso 322450



Original Predicción de la Red Predicción Médica

Figura 19: Comparativa visual del caso 322450

		True diagnosis		
		Negative	Positive	Total
Screening test	Negative	13405	229	13634
	Positive	949	2055	3004
	Total	14354	2284	16638

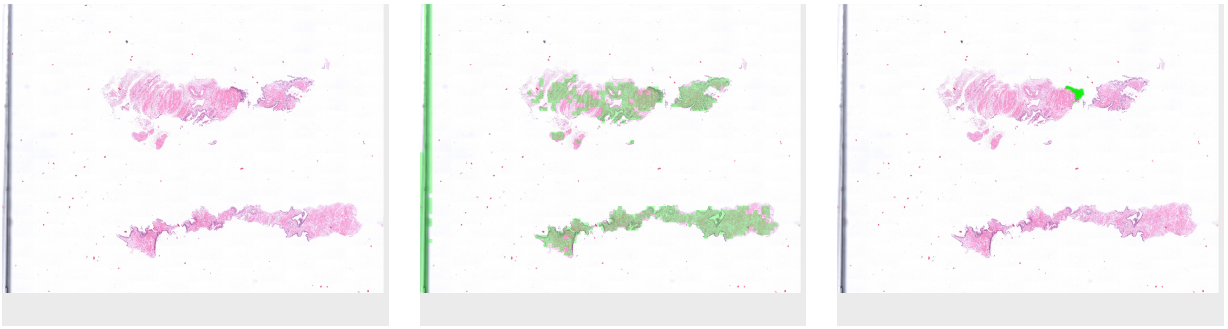
Tabla 5: Matriz de confusión del caso 322454

Original Predicción de la Red Predicción Médica

Figura 20: Comparativa visual del caso 322454

		True diagnosis		
		Negative	Positive	Total
Screening test	Negative	6804	7	6811
	Positive	604	11	615
	Total	7408	18	7426

Tabla 6: Matriz de confusión del caso 322444

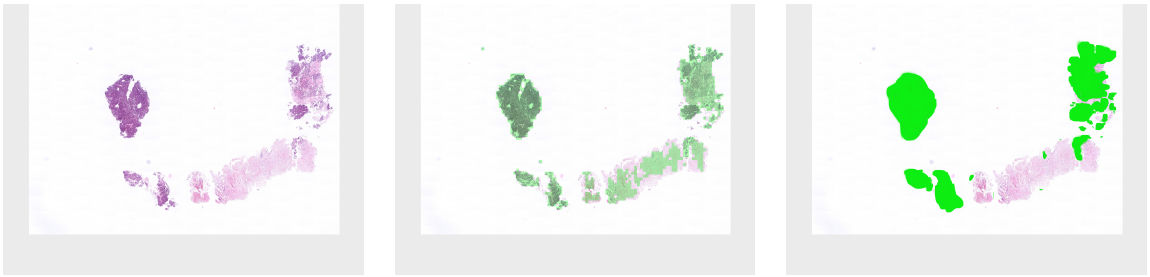


Original Predicción de la Red Predicción Médica

Figura 21: Comparativa visual del caso 322444

		True diagnosis		
		Negative	Positive	Total
Screening test	Negative	10768	138	10904
	Positive	244	602	846
	Total	11012	738	11750

Tabla 7: Matriz de confusión del caso 315965



Original Predicción de la Red Predicción Médica

Figura 22: Comparativa visual del caso 315965

Se puede observar, a partir de las matrices de confusión, que el resultado del aprendizaje es prometedor debido a la pequeña cantidad de falsos negativos, quedando como objetivo futuro la reducción de los falsos positivos y falsos negativos con el objetivo de mejorar la precisión en el diagnóstico, facilitando así la labor de los profesionales de la medicina.

8 Conclusiones y trabajos futuros

Tras la realización del proyecto, hemos conseguido aprender el funcionamiento de la redes neuronales dentro de los algoritmos de aprendizaje máquina. Además hemos instalado y utilizado un *framework* como MXNet para realizar varias pruebas de redes neuronales.

Adicionalmente hemos configurado una máquina para el cómputo masivo que requiere un algoritmo de este estilo, añadido dos tarjetas gráficas, e instalado todos los requerimientos para el funcionamiento del sistema.

Tras la instalación del *framework* MXNET, hemos conseguido entrenar y probar correctamente los modelos MNIST y CIFAR con la configuración recomendada por este, obteniendo un 98.1 % y 93.1 % de *accuracy* respectivamente tras su fase de entrenamiento.

Nos hemos enfrentado a un caso real y generado una metodología a seguir para conseguir un modelo capaz de funcionar en dicho caso. Esta metodología puede ser usada para conseguir otros modelos en otros tipos de imágenes con características similares.

Para el futuro quedan varios trabajos entre ellos podemos destacar dos muy importantes: conseguir un *accuracy* superior al 95 % pudiendose utilizar en situaciones reales, y reducir el tiempo de inferencia a tiempo real.

Para el primer objetivo se tienen que realizar algunos estudios, desde el tamaño de los *tiles* hasta el resto de parámetros de la red. Todos estos parámetros deben ser probados de manera experimental, se basan en ensayo y error.

Para reducir el tiempo de inferencia se podrían aplicar técnicas de paralelismo, y optimización del código usado, a la vez que una implementación en un lenguaje compilado en lugar de uno interpretado.

En el caso de querer realizar un procesamiento posterior y no solo detectar las zonas cancerígenas, podría usarse una RNN que a partir de los resultados obtenidos de la red desarrollada, con el objetivo de obtener un diagnóstico completo desde la biopsia.

Referencias

[1] Antonio Matencio Escolar. Evaluación de algoritmos deep learning para procesadores intel ® xeon, 6 2017.

[2] José Antonio Bernabé Díaz. Acelerando la meta-heurísstica de deep learning por medio de técnicas de hpc usando el xeon phi, 6 2016.

[3] Donald O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, New York, June 1949.

[4] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 44(1.2):206–226, Jan 2000.

[5] Stanford University. Stanford Electronics Laboratories, B. Widrow, United States. Office of Naval Research, United States. Army Signal Corps, United States. Air Force, and United States. Navy. *Adaptive "adaline"neuron using chemical "memistors."*. 1960.

[6] M.L. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. Expanded, 1988.

[7] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[8] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[9] Mustafa Coşkun, Huseyin Guruler, Ayhan Istanbullu, and Musa Peker. Determining the appropriate amount of anesthetic gas using dwt and emd combined with neural network. 39:1=10, 02 2015.

[10] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *CoRR*, abs/1703.09039, 2017.

[11] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

[12] Vivienne Sze, Yu Hsin Chen, Tien Ju Yang, and Joel S. Emer. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[13] Saleh Albelwi and Ausif Mahmood. A framework for designing the architectures of deep convolutional neural networks. *Entropy*, 19(6), 2017.

[14] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

[15] Michael A. Nielsen. Neural networks and deep learning. http://neuralnetworksanddeeplearning.com/, 2018.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[17] Pavlo M. Radiuk. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. 20, 12 2017.

[18] Intel. Openvino toolkit. https://software.intel.com/en-us/openvino-toolkit. Accessed: 2018-01-28.

[19] Amazon. Apache mxnet en aws. https://aws.amazon.com/es/mxnet/. Accessed: 2018-06-13.

[20] Haojin Yang, Martin Fritzsche, Christian Bartz, and Christoph Meinel. Bmxnet: An open-source binary neural network implementation based on mxnet. *CoRR*, abs/1705.09864, 2017.

[21] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. *CoRR*, abs/1502.02551, 2015.

[22] Shujing Zhang, Bo He, Rui Nian, Jing Wang, Bo Han, Amaury Lendasse, and Guang Yuan. Fast image recognition based on independent component analysis and extreme learning machine, 09 2014.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[24] Maciej A. Mazurowski, Mateusz Buda, Ashirbani Saha, and Mustafa R. Bashir. Deep learning in radiology: an overview of the concepts and a survey of the state of the art. *CoRR*, abs/1802.08717, 2018.

[25] Hideharu Ohsugi, Hitoshi Tabuchi, Hiroki Enno, and Naofumi Ishitobi. Accuracy of deep learning, a machine-learning technology, using ultra-wide-field fundus ophthalmoscopy for detecting rhegmatogenous retinal detachment. *Scientific Reports*, 7(1):9425, 2017.

[26] Angel Cruz-Roa, Hannah Gilmore, Ajay Basavanhally, Michael Feldman, Shridar Ganesan, Natalie N. C. Shih, John Tomaszewski, Fabio A. González, and Anant Madabhushi. Accurate and reproducible invasive breast cancer detection in whole-slide images: A deep learning approach for quantifying tumor extent. *Scientific Reports*, 7:46450 EP –, Apr 2017. Article.

[27] Piotr Chudzik, Somshubra Majumdar, Francesco Calivá, Bashir Al-Diri, and Andrew Hunter. Microaneurysm detection using fully convolutional neural networks. *Computer Methods and Programs in Biomedicine*, 158:185 – 192, 2018.

[28] Christof Angermueller, Tanel Pärnamaa, Leopold Parts, and Oliver Stegle. Deep learning for computational biology. *Molecular Systems Biology*, 12(7), 2016.

[29] O. Tange. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47, Feb 2011.

[30] Geert Litjens, Clara I. Sánchez, Nadya Timofeeva, Meyke Hermsen, Iris Nagtegaal, Iringo Kovacs, Christina Hulsbergen - van de Kaa, Peter Bult, Bram van Ginneken, and Jeroen van der Laak. Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. *Scientific Reports*, 6(1):26286, 2016.

Anexos

I Instalación del Sistema

Se ha reinstalado una máquina con hardware adicional para el desarrollo de este trabajo. Esta dispone de: una CPU Intel(R) Xeon(R) CPU E5-2603 V3 @1.6GHz, 64GB de Memoria DDR4 a 2133MHz.

Posteriormente se le añadieron dos tarjetas gráficas: Nvidia GTX 1080 Pascal, y Nvidia GTX 980 Maxwell, y adicionalmente un SSD Samsung Evo 500GB. Dicho SSD ha necesitado un adaptador físico para poder ser introducido físicamente en la máquina.

I.I Sistema Operativo

Recientemente la máquina fue reinstalada usando el sistema operativo Centos 7, el cual es la versión comunitaria de RedHat Enterprise Linux, un sistema creado para actividades profesionales por su seguridad y estabilidad.

I.II Servicios

Se han instalado los siguientes servicios para añadir la máquina al *cluster* actual: Nis, Nfs, y Slurm.

Nis se instaló con el proposito de tener todos los usuarios centralizados. Nfs se utiliza para compartir los archivos entre las distintas máquinas y el punto de entrada. Slurm se ha usado como sistema colas para los trabajos, pudiendose ejecutar estos en exclusión mutua.

I.III MXNet y sus dependencias

Se va a usar tarjetas gráficas Nvidia para acelerar el cómputo, en concreto del *training* de la red neuronal, por lo que se instalará Cuda usando las instrucciones oficiales de Nvidia.

Tras instalar Cuda, se necesitará Cudnn, un conjunto de librerias para redes neuronales basado en Cuda, y al igual que este se seguirá el manual oficial de Nvidia.

Como se usará el *binding* de Pyhton, es necesario instalar el interpetré de Python y el autoinstalador Pip. Siguiendo las instrucciones de MXNet, se usará Pip para instalar MXNet y todas sus dependencias.

II Training MNIST

Epoch	Training	Validation
0	0.955236	0.963774
1	0.970439	0.965267
2	0.982264	0.965167
3	0.981841	0.966760
4	0.985220	0.975318
5	0.985220	0.972432
6	0.988598	0.973925
7	0.991554	0.975020
8	0.994088	0.969845
9	0.992399	0.975418
10	0.998733	0.982484
11	0.998733	0.982186
12	0.999155	0.982385
13	0.999155	0.982385
14	0.999155	0.982086
15	0.999155	0.982285
16	0.999155	0.982186
17	0.999155	0.982086
18	0.999155	0.981887
19	0.999155	0.981986

Tabla 8: Tabla del accurary del ejemplo del MNist por épocas

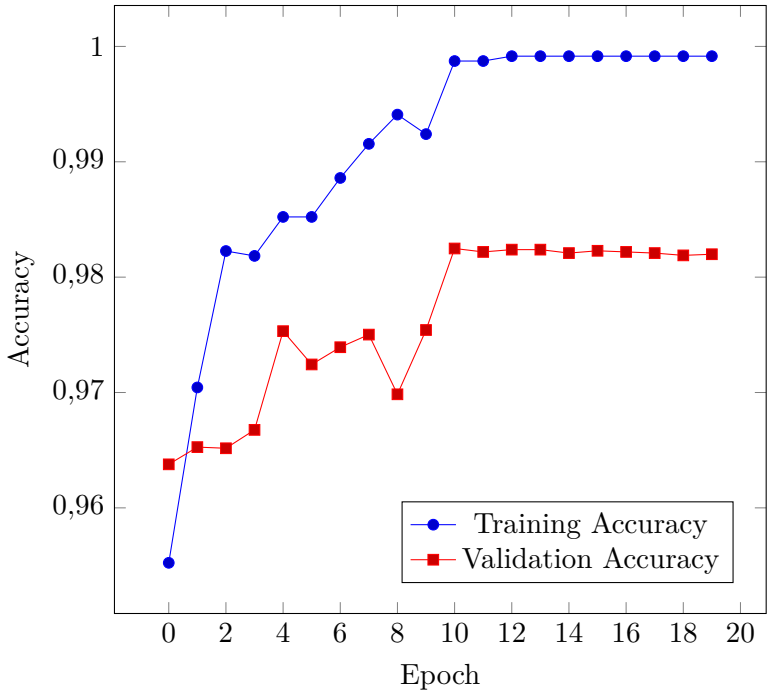


Figura 23: Gráfica del accurary del ejemplo del MNist por épocas

III Training CIFAR

Epoch	Training	Validation	Epoch	Training	Validation	Epoch	Training	Validation
0	369.531	409.415	60	902.344	882.512	120	931.250	875.000
1	546.094	547.075	61	901.042	852.364	121	930.469	889.523
2	611.111	604.868	62	910.156	878.806	122	929.688	893.630
3	678.906	703.526	63	920.139	860.877	123	949.219	889.523
4	714.844	745.793	64	911.719	880.538	124	930.469	891.827
5	709.201	774.239	65	913.281	878.806	125	926.215	892.628
6	756.250	767.328	66	907.118	882.512	126	941.406	884.215
7	758.681	792.768	67	910.937	876.302	127	936.632	878.506
8	771.094	778.283	68	916.406	843.550	128	939.844	887.362
9	813.281	804.087	69	907.986	886.719	129	927.344	894.331
10	782.986	790.966	70	919.531	878.405	130	943.576	891.927
11	807.813	805.889	71	924.479	871.995	131	931.250	902.444
12	817.187	807.292	72	918.750	869.561	132	927.344	890.425
13	822.917	834.135	73	917.188	885.817	133	939.236	862.280
14	811.719	837.240	74	909.722	874.499	134	946.094	895.633
15	826.389	826.022	75	928.125	887.720	135	936.632	893.029
16	822.656	833.465	76	918.750	875.701	136	936.719	891.515
17	845.313	841.546	77	916.667	887.119	137	934.375	885.717
18	835.069	823.017	78	915.625	883.714	138	938.368	887.520
19	853.125	850.761	79	924.479	881.911	139	937.500	886.218
20	843.750	835.537	80	924.219	869.165	140	933.594	898.738
21	833.333	848.858	81	917.188	882.111	141	943.576	889.423
22	842.187	859.976	82	912.326	881.410	142	941.406	891.827
23	856.771	840.445	83	929.688	875.000	143	949.653	884.615
24	857.812	847.310	84	913.281	891.126	144	929.688	883.999
25	857.031	849.459	85	927.083	878.806	145	941.406	891.026
26	879.340	854.167	86	922.656	882.712	146	934.896	891.026
27	846.094	858.474	87	907.986	882.913	147	939.063	899.339
28	863.281	854.567	88	923.438	868.078	148	939.063	887.921
29	875.000	867.788	89	925.781	877.103	149	941.840	893.429
30	875.000	861.178	90	915.799	876.903	150	941.406	893.630
31	874.132	864.183	91	926.562	875.300	151	947.049	887.620
32	864.844	864.616	92	921.875	881.310	152	944.531	888.845
33	889.844	872.396	93	923.611	879.307	153	939.844	873.498
34	879.340	870.092	94	919.531	879.507	154	940.972	889.724
35	893.750	855.569	95	910.590	878.205	155	939.844	883.313
36	888.281	868.389	96	933.594	876.978	156	937.500	890.224
37	894.965	867.688	97	926.562	887.420	157	934.896	879.908
38	891.406	845.353	98	914.062	885.517	158	942.969	895.833
39	878.472	856.771	99	932.813	892.829	159	940.972	901.943
40	888.281	847.903	100	925.781	886.218	160	933.594	893.493
41	883.594	877.905	101	924.479	880.909	161	942.187	894.631
42	884.549	846.254	102	926.562	883.614	162	940.104	849.459
43	900.781	850.561	103	924.479	879.107	163	935.156	884.215
44	875.000	854.968	104	923.438	886.669	164	942.187	890.224
45	907.118	863.381	105	934.375	890.325	165	930.556	881.010
46	905.469	870.192	106	939.236	869.591	166	942.969	897.636
47	888.889	875.901	107	939.844	891.326	167	934.896	879.708
48	900.781	871.737	108	930.469	871.995	168	938.281	895.273
49	902.344	869.792	109	927.951	888.421	169	930.469	895.533
50	905.382	873.698	110	927.344	893.830	170	933.160	887.821
51	906.250	881.611	111	920.139	885.917	171	935.937	891.426
52	910.937	861.078	112	920.312	890.131	172	953.125	895.533
53	907.118	878.806	113	946.094	870.793	173	941.840	892.127
54	903.906	869.692	114	935.764	888.822	174	949.219	896.735
55	922.743	867.688	115	937.500	892.728	175	935.764	894.932
56	903.906	857.002	116	940.625	893.930	176	937.500	893.295
57	914.844	868.890	117	935.764	892.528	177	944.531	903.145
58	907.986	881.310	118	925.781	894.231	178	946.181	888.121
59	905.469	871.695	119	918.403	886.018	179	948.438	885.817

Epoch	Training	Validation	Epoch	Training	Validation	Epoch	Training	Validation	Epoch	Training	Validation
180	946.875	896.434	210	989.583	929.688	240	992.969	931.369	270	996.094	931.390
181	931.424	896.935	211	986.719	930.489	241	989.062	929.387	271	993.056	931.490
182	939.063	896.835	212	991.406	929.988	242	988.715	932.492	272	997.656	930.676
183	946.181	898.638	213	983.507	930.489	243	993.750	929.387	273	997.656	930.990
184	945.312	890.625	214	985.156	930.088	244	989.844	931.190	274	994.792	931.991
185	944.531	897.636	215	984.375	929.087	245	991.319	929.187	275	995.313	931.691
186	934.896	879.507	216	991.406	930.083	246	986.719	928.586	276	996.875	932.292
187	941.406	894.531	217	989.062	930.088	247	990.451	931.891	277	995.660	930.689
188	935.156	884.615	218	987.847	930.689	248	990.625	928.699	278	996.875	931.090
189	934.028	885.717	219	991.406	930.088	249	989.844	930.188	279	996.528	931.190
190	945.312	899.139	220	994.531	930.088	250	992.188	930.789	280	994.531	931.468
191	947.049	896.134	221	981.771	930.088	251	994.531	930.990	281	996.875	931.390
192	944.531	889.834	222	988.281	930.990	252	995.313	931.791	282	999.132	930.889
193	946.875	901.843	223	987.847	932.392	253	994.792	931.591	283	997.656	931.691
194	950.521	893.730	224	984.375	930.973	254	993.750	932.792	284	996.094	931.290
195	941.406	893.029	225	989.844	929.988	255	996.528	931.891	285	997.396	932.091
196	942.187	890.825	226	992.188	929.087	256	995.313	929.885	286	996.875	932.192
197	945.312	882.111	227	990.625	928.686	257	992.188	932.292	287	993.056	932.091
198	948.438	899.840	228	986.719	932.091	258	996.528	931.591	288	997.656	931.566
199	953.993	893.730	229	992.188	931.290	259	995.313	931.490	289	995.313	932.091
200	965.625	921.974	230	989.062	930.789	260	996.875	932.192	290	996.528	931.190
201	978.906	925.381	231	991.319	929.988	261	994.792	932.192	291	996.875	931.891
202	977.431	926.282	232	987.500	930.083	262	996.875	932.392	292	999.219	932.592
203	989.062	927.284	233	993.750	928.486	263	994.792	931.390	293	993.056	931.791
204	980.469	927.784	234	991.319	931.090	264	993.750	931.566	294	994.531	931.290
205	971.354	928.986	235	989.062	930.288	265	996.094	931.991	295	994.792	931.290
206	986.719	929.087	236	993.750	928.686	266	997.396	931.791	296	998.437	931.764
207	982.639	928.385	237	994.792	932.091	267	992.969	931.490	297	996.094	932.692
208	985.156	928.896	238	995.313	931.090	268	997.656	931.390	298	999.132	932.192
209	980.469	929.988	239	995.660	929.187	269	994.792	931.390	299	995.313	931.791

Tabla 9: Tabla del accurary del ejemplo del Cifar por épocas

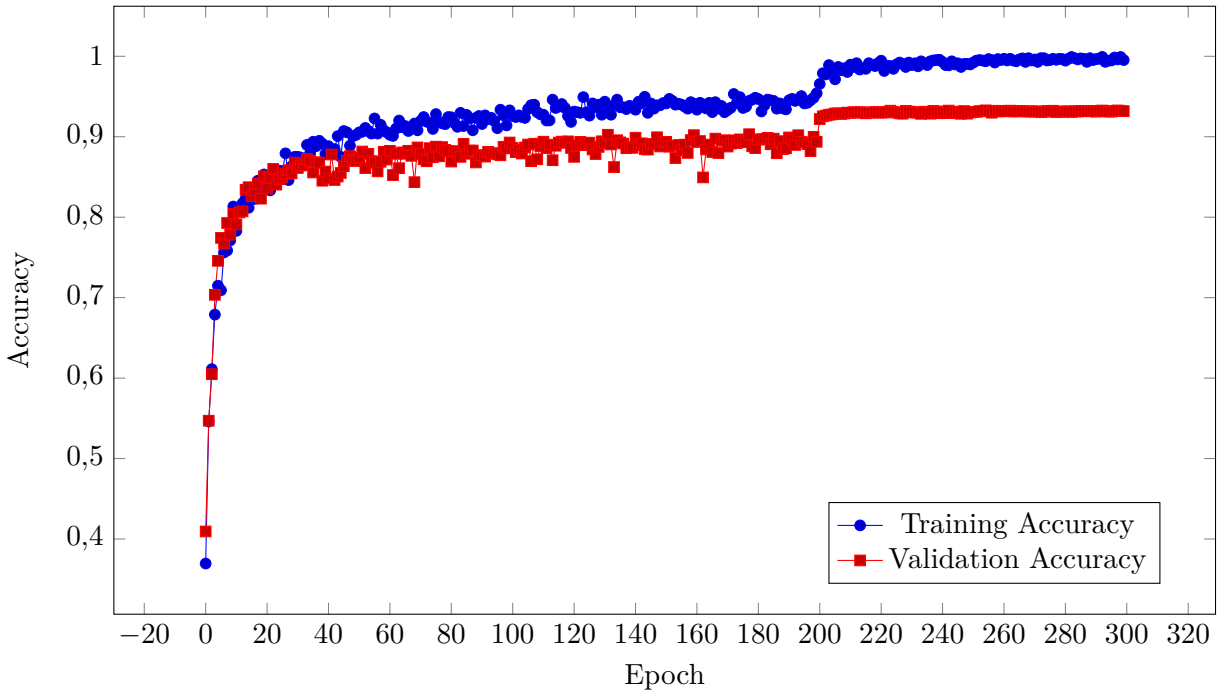


Figura 24: Gráfica del accurary del ejemplo del Cifar por épocas