Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Sistema Basado en Deep Learning Para la Identificación de Estilos de Cómics

Alumno: Carlos Teba Hermoso

Tutor: Prof. D. Francisco Charte Ojeda
Prof. Dña. María Dolores Pérez Godoy

Dpto: Informática

Septiembre, 2019

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Francisco Charte Ojeda y Doña María Dolores Pérez Godoy, tutores del Trabajo de Fin de Grado titulado: *Sistema basado en Deep Learning para la identificación de estilos de cómics*, que presenta Carlos Teba Hermoso, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Agosto de 2019

El alumno:

TEBA HERMOSO CARLOS - 77366703Q
Firmado digitalmente por TEBA HERMOSO CARLOS - 77366703Q
Fecha: 2019.08.27 17:52:04 +02'00'

Carlos Teba Hermoso

Los tutores:

CHARTE OJEDA FRANCISCO - 25996290B
Firmado digitalmente por CHARTE OJEDA FRANCISCO - 25996290B
Fecha: 2019.08.28 09:22:07 +02'00'

Francisco Charte Ojeda

PEREZ GODOY MARIA DOLORES - 75093149N
Firmado digitalmente por PEREZ GODOY MARIA DOLORES - 75093149N
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-75093149N, givenName=MARIA DOLORES, sn=PEREZ GODOY, cn=PEREZ GODOY MARIA DOLORES - 75093149N
Fecha: 2019.08.30 19:59:53 +02'00'

María Dolores Pérez Godoy

Índice

Índice de figuras

Índice de tablas

1. Introducción, motivación y objetivos del proyecto

1.1. Introducción

En este capítulo se presentan la motivación que da lugar a la realización de este Trabajo de Fin de Grado, los objetivos que se pretenden alcanzar, la metodología seguida para alcanzarlos, así como la estructura de este documento y la temporización del trabajo realizado.

1.2. Motivación

Uno de los términos más populares de los últimos años es el de **ciencia de datos** (*data science,* en inglés), hasta el punto de que muchos medios lo han llamado "el trabajo más atractivo del siglo XXI" (*'The sexiest job of the 21st century'*). Esta ciencia de datos es un área multidisciplinar cuyo principal objetivo es obtener conocimiento a partir de datos. A pesar de esta moderna popularidad de la ciencia de datos, ya existía desde los años 90 un concepto similar, denominado **Proceso de Descubrimiento de Conocimiento en Bases de Datos** (KDD, por sus siglas en inglés).

Además de los nombres Ciencia de Datos y KDD, existen otros, como Minería de Datos, Aprendizaje Automático (*Machine Learning*), que a menudo se confunden y se utilizan indistintamente, aunque hagan referencia a técnicas, métodos o campos de estudio diferentes, pero muy relacionados unos con otros.

De todos estos términos, el más frecuente últimamente es el de *Deep Learning* (Aprendizaje Profundo en español, aunque apenas se usa, prefiriéndose habitualmente la versión en inglés), debido a las últimas aplicaciones en coches autónomos, aplicaciones como FaceApp [29] o AlphaGo [30].

Como motivación personal, este campo de la ciencia de datos, en particular el *Machine Learning* y el *Deep Learning*, presentan un reto interesante, además de muchas opciones de trabajo. Otro aspecto atrayente de este proyecto es el tema de las imágenes analizadas. Hasta donde sabemos, todas las aplicaciones actuales de reconocimiento de imágenes se aplican a fotografías de elementos "reales", como detección de vehículos, reconocimiento facial, identificación de letras o dígitos, etc. No existen, que sepamos, aplicaciones que traten de identificar autores o, en nuestro caso, estilos de creaciones artísticas, como obras musicales o ilustraciones.

1.3. Objetivos

En este Proyecto se pretenden alcanzar los siguientes objetivos:

- Obtención de una visión general del proceso de descubrimiento de conocimiento a partir de bases de datos y de métodos de minería de datos.
- Estudio de métodos supervisados para clasificación de patrones, incluyendo técnicas de aprendizaje profundo y, en particular, CNNs.
- Identificación de distintos estilos de cómics y recopilación de un conjunto de imágenes de cada estilo, generando la base de datos que servirá para llevar a cabo el aprendizaje.
- Selección del *framework* adecuado para llevar a cabo el desarrollo del sistema, de entre los existentes en la actualidad: TensorFlow, Keras, MXNET, etc.
- Implementación del sistema empleando el *framework* antes seleccionado, entrenamiento con la base de datos de imágenes previamente recopiladas y análisis de su rendimiento.
- Redacción de la memoria correspondiente del TFG.

1.4. Metodología

Para la realización de este proyecto se seguirá la siguiente metodología:

- Revisión bibliográfica acerca del procedimiento de descubrimiento de conocimiento a partir de bases de datos (*Knowledge Discovery in Databases, KDD*), métodos de aprendizaje en general y redes neuronales en particular.
- Estudio de técnicas de aprendizaje profundo (*Deep Learning, DL*) y el funcionamiento de las redes neuronales convolucionales.
- Análisis de requisitos del sistema a desarrollar, incluyendo la generación de una base de datos apropiada para su funcionamiento.
- Recopilación, preprocesamiento y etiquetado de la base de datos de imágenes.
- Evaluación de las distintas alternativas para la creación de redes neuronales convolucionales (*Convolutional Neural Networks, CNNs*) y selección justificada de la que se utilizará.
- Diseño e implementación del sistema empleando la tecnología elegida previamente.

- Entrenamiento del sistema con la base de datos de imágenes anteriormente generada y análisis del rendimiento predictivo del mismo.
- Realización de pruebas.
- Redacción de la memoria del proceso de diseño, implementación y evaluación del sistema.

1.5. Estructura del documento

A continuación, se presenta una breve descripción de los capítulos en que se divide el presente documento y el contenido de los mismos.

En el **primer** y actual **capítulo** se da una visión general del proyecto, se presentan los objetivos que pretendemos alcanzar y la metodología seguida para ello.

En los dos siguientes capítulos se exponen las bases teóricas del proyecto. En el **Capítulo 2** se presenta el Proceso de Descubrimiento de Conocimiento en Bases de Datos, el proceso estructurado a seguir para el aprendizaje a partir de datos. Posteriormente, en el **Capítulo 3**, se exponen los fundamentos de las Redes Neuronales Artificiales y las Redes Neuronales Convolucionales.

A continuación, los dos siguientes capítulos presentan los diseños de la base de datos y del sistema de aprendizaje utilizados en este proyecto. Primeramente, en el **Capítulo 4** se expone el diseño, los procesos de recopilación y preprocesamiento y la implementación de la Base de Datos utilizada. Por otra parte, en el **Capítulo 5** se presentan las condiciones que ha de cumplir el sistema de aprendizaje, así como las arquitecturas y tecnologías utilizadas.

Seguidamente, en el **Capítulo 6** se detallan los experimentos realizados, así como la justificación del proceso seguido para alcanzar el modelo final entrenado con la base de datos.

En el **Capítulo 7** se recopilan las conclusiones obtenidas tras la realización de este proyecto y se proponen algunas guías para un trabajo futuro.

En último lugar, se incluyen la **Bibliografía** utilizada y **anexos** para la instalación y configuración de entornos de desarrollo para sistemas Deep Learning, en local (**Anexo I**) y en la nube (**Anexo II**), así como un detalle de los ficheros que forman la base de datos y el código del proyecto (**Anexo III**).

1.6. Temporización del trabajo

Este Trabajo de Fin de Grado se ha realizado a lo largo del curso 2018/2019, junto al resto de asignaturas del último curso de la titulación. Por lo tanto, la carga de trabajo no se ha repartido uniformemente a lo largo del curso, sino que se ha realizado un mayor esfuerzo en aquellos meses o épocas en los que las asignaturas demandaban menos tareas y reduciéndose este esfuerzo en las últimas semanas de los cuatrimestres y en épocas de exámenes. A continuación, se presenta un sumario del trabajo realizado a lo largo del curso.

La primera tarea realizada fue la de la **revisión de la base teórica** necesaria. Esta se compone principalmente de una revisión del Proceso de Descubrimiento de Conocimiento en Bases de Datos, utilizando los apuntes de la asignatura Minería de Datos [27] y el libro *Introducción a la Minería de Datos*, de José Hernández [2]. La segunda parte de esta tarea consiste en el estudio de las técnicas y la metodología del Aprendizaje Profundo (*Deep Learning*), utilizando los cursos de la *Deep Learning Specialization*, de Andrew NG, en la plataforma Coursera [28] y el libro *Deep Learning*, de Ian Goodfellow [1]. Esta tarea se realizó durante los meses de **Septiembre y Octubre**.

A continuación, durante los meses de **Noviembre y Diciembre**, se llevó a cabo la revisión de los estilos de cómics y la **creación de la Base de Datos**, descritas en el capítulo 4.

Una vez acabados los exámenes de Enero, se realizó el estudio y análisis de los distintos **entornos y tecnologías** para desarrollar un proyecto ***Deep Learning***, así como de las distintas **arquitecturas** de Redes Neuronales Convolucionales disponibles. Esta tarea ocupó los meses de **Febrero**, **Marzo** y buena parte de **Abril**.

Finalmente, desde **Junio hasta mediados de Agosto**, se han realizado los diferentes **experimentos** descritos en el capítulo 6, la extracción de las **conclusiones** a partir de estos y la **redacción de la memoria** del TFG.

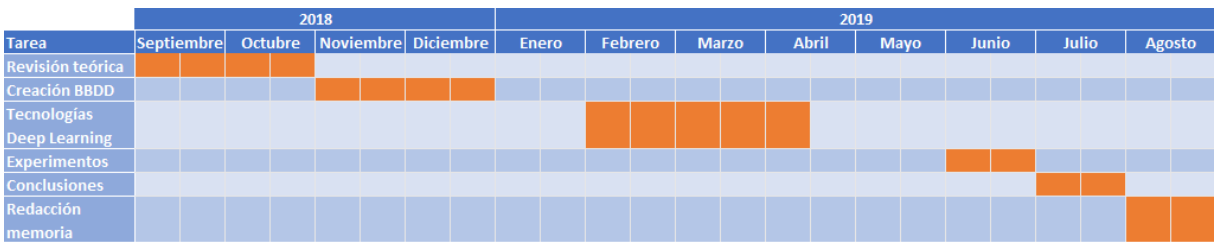


Ilustración 1 Cronograma del proyecto (Elaboración propia)

2. El Proceso de Descubrimiento de Conocimiento en Bases de Datos y la Minería de Datos

En este capítulo se presenta el proceso utilizado para la obtención de conocimiento a partir de datos, el Proceso de Descubrimiento de Conocimiento en Bases de Datos, así como las fases de las que se compone.

2.1. El proceso de descubrimiento de conocimiento en bases de datos (KDD)

Aunque a menudo se utiliza el término Minería de Datos para denominar este proceso de extracción de patrones a partir de datos, existe otro término utilizado para referirse a este proceso, el de “descubrimiento de conocimiento en bases de datos” (*Knowledge Discovery in Databases, KDD)*. La principal diferencia entre estos dos términos es que el término KDD hace referencia a un proceso compuesto por una serie de fases, en el que la minería de datos es una de esas fases.

Se define el KDD [Fayyad et al. 1996a] como “el proceso no trivial de identificar patrones válidos, novedosos, potencialmente útiles y comprensibles a partir de los datos”. A partir de esta definición se extraen cuáles han de ser las propiedades esperadas del conocimiento extraído:

- **Válido**: los patrones obtenidos han de seguir siendo válidos para nuevos datos, y no solamente para aquellos utilizados para la obtención de los patrones.
- **Novedoso**: debe aportar algo nuevo para el sistema y para el usuario.
- **Potencialmente útil**: debe conducir a acciones que reporten algún beneficio al usuario.
- **Comprensible**: patrones que no sean comprensibles hacen muy difícil o imposible la interpretación y el uso de estos patrones en la toma de decisiones.

El KDD es un proceso complejo, que tiene como objetivo la extracción de patrones en los datos (minería de datos) así como la evaluación e interpretación de estos patrones.

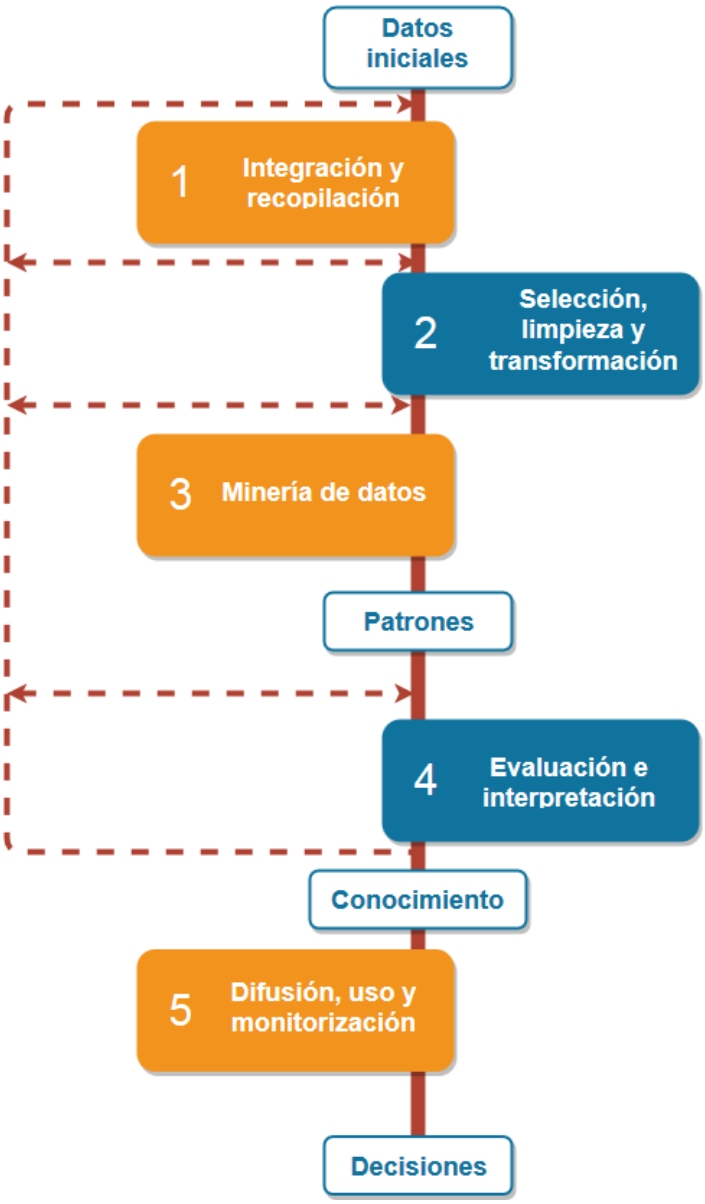


Ilustración 2 Proceso de extracción de conocimiento de bases de datos, KDD. (Fuente: Elaboración propia).

El KDD es un proceso **iterativo** ya que a la salida de algunas de las fases puede que tengamos que volver a pasos anteriores y es habitual tener que realizar varias iteraciones para extraer conocimiento de alta calidad. También es un proceso **interactivo** ya que un experto en el dominio del problema debe ayudar en las distintas fases del proceso.

El proceso KDD se divide en 5 fases, tal y como se ve en la Ilustración 2. En la fase de **integración y recopilación** se seleccionan las fuentes de información útiles y se unifican todos los datos a un formato común, resolviendo inconsistencias. En la fase de **selección, limpieza y transformación** de los datos se corrigen los datos incorrectos y se seleccionan aquellas instancias y atributos para facilitar la siguiente

tarea. En la fase de **minería de datos** de decide la tarea a realizar, el método o algoritmo a utilizar. En la fase de **evaluación e interpretación** los expertos evalúan los patrones obtenidos y, si es necesario, se vuelve a fases anteriores. En la última fase de **difusión, uso y monitorización** se hace uso del nuevo conocimiento.

A continuación, se describen brevemente las distintas fases del proceso KDD.

2.1.1. Integración y recopilación

Normalmente, los datos necesarios para el proceso KDD no provienen de una única fuente de datos, sino de diferentes fuentes (distintas organizaciones, bases de datos públicas, datos recabados expresamente para el problema, etc.). Esto presenta un obstáculo, ya que cada base de datos utiliza distintos formatos, distintos tipos de claves primarias, distintas unidades de medida para magnitudes como el tiempo, la masa, la velocidad, etc., que hay que homogeneizar, distintos tipos de error, etc. Así pues, el primer paso, una vez recopilados los datos, es la integración de todos estos datos en un esquema de datos unificado.

2.1.2. Selección, limpieza y transformación

Para extraer conocimiento de calidad no solamente necesitamos un buen algoritmo de minería de datos, sino que los datos utilizados también han de tener una calidad alta. Además, muchos de estos algoritmos de extracción de patrones tienen restricciones en relación a los datos con los que pueden trabajar (por ejemplo, algoritmos que únicamente pueden trabajar con datos en formato numérico y no con cadenas de caracteres, redes convolucionales que requieren que las imágenes tengan un tamaño específico en píxeles, etc.).

En consecuencia, una vez recopilados los datos, el siguiente paso es preparar y seleccionar el conjunto de datos que utilizará el algoritmo de minería de datos.

2.1.2.1. Limpieza

En los datos pueden existir valores que no se ajustan al comportamiento general (conocidos como datos anómalos u *outliers*). Estas muestras anómalas pueden ser datos erróneos, o datos correctos que sencillamente son diferentes a los demás. El tratamiento a realizar sobre estos outliers dependerá del algoritmo con el que vayamos a trabajar: algunos algoritmos ignoran estos *outliers*, mientras que otros son muy sensibles a ellos. También puede darse el caso de que estos datos extraños sean más interesantes que los datos que siguen el patrón general (por ejemplo, compras por un

importe muy superior al de las compras habituales en un análisis de fraude con tarjetas de crédito).

Otro problema que podemos tener es que nos falten datos o valores (*missing values*). Estos valores perdidos pueden deberse a muchas causas y existen gran cantidad de métodos para tratar con ellos. Habitualmente necesitaremos conocimiento experto sobre el dominio del problema para elegir el método o métodos a utilizar.

Por otro lado, en la etapa de limpieza también se trata con la presencia de "ruido", un concepto distinto al de "*outlier*". El ruido es presencia de datos incorrectos por errores en equipos de medida, transcripción o algún otro fallo, mientras que los *outliers* son datos correctos pero que no siguen la distribución de la mayoría del conjunto de datos.

2.1.2.2. Selección

Otro problema frecuente es la selección de atributos relevantes para el problema, ya que no todos los atributos disponibles son igual de importantes para la extracción de conocimiento. Por ejemplo, si estamos analizando datos médicos para predecir si un paciente tendrá o no cáncer, el historial clínico del paciente seguramente sea importante mientras que el DNI del paciente no lo sea.

También es posible que tengamos que seleccionar algunas de las tuplas en lugar de utilizar todos los datos disponibles (muestreo de datos), ya que puede que no dispongamos de una máquina con la capacidad de cómputo necesaria para trabajar con todos los datos disponibles o bien haya muestras de datos que no aporten información relevante para la resolución del problema

2.1.2.3. Transformación

Otra tarea es la transformación de los datos a otros tipos o unidades con los que el algoritmo pueda trabajar mejor. La tarea de **construcción de atributos** consiste en construir nuevos atributos aplicando alguna función sobre los atributos originales. Por ejemplo, puede ser que construir un atributo de densidad de población nos resulte más útil que utilizar los atributos de población y área por separado.

También es habitual tener que modificar el tipo de datos para adecuarse a los tipos con los que puede trabajar el algoritmo. Si tenemos un atributo que indique, por ejemplo, el tipo de vehículo podemos ***numerizar*** ese atributo, asignando un número a cada tipo de vehículo.

Por otro lado, podemos **discretizar** atributos continuos según una serie de intervalos.

2.1.3. Minería de datos

En esta fase el objetivo es la utilización de un modelo basado en los datos para producir nuevo conocimiento. Antes de elegir qué modelo utilizar hay que determinar una serie de características de este:

- Tipo de tarea de minería de datos.
- Elegir el tipo de modelo para la tarea elegida.
- Seleccionar un algoritmo que resuelva la tarea utilizando el tipo de modelo que nos interesa.

A continuación, se describen las opciones más habituales en estas características del modelo.

2.1.3.1. Tareas

Las tareas de minería de datos (o tareas de aprendizaje automático) son distintos tipos de problemas a ser resueltos por el algoritmo de minería de datos. Cada tarea obtiene un tipo de conocimiento que puede ser muy distinto al de otras tareas y tiene una serie de requisitos que también pueden ser muy diferentes entre tareas.

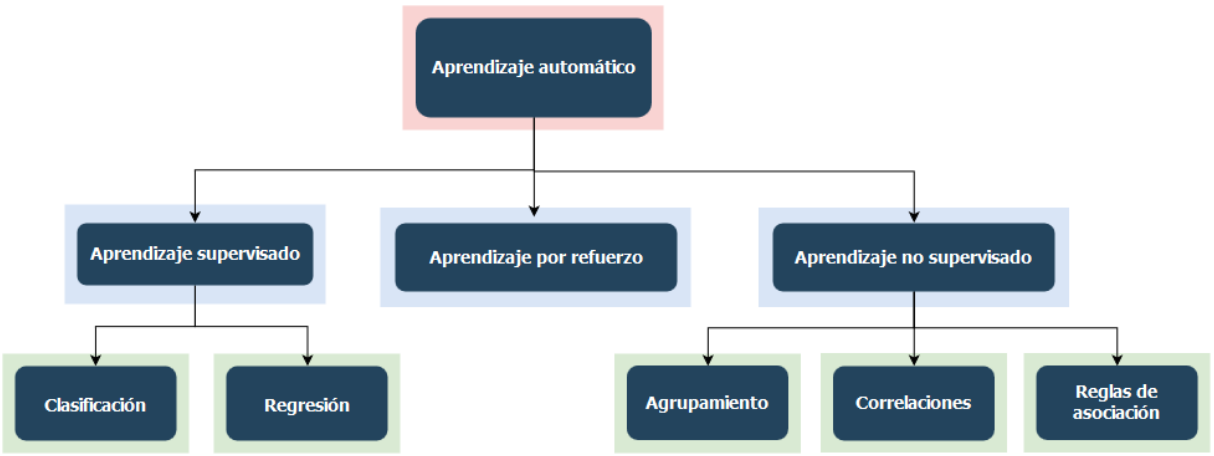


Ilustración 3 Taxonomía de tareas de aprendizaje automático. (Fuente: Elaboración propia)

Las tareas se suelen dividir en **supervisadas**, **no supervisadas** o **por refuerzo**. Las tareas más habituales en minería de datos son:

- **Tareas supervisadas**: se presenta al sistema una serie de ejemplos etiquetados, es decir, un conjunto de datos de entrada y sus correspondientes

salidas, con el objetivo de aprender reglas que hagan corresponder entradas a salidas.

- **Clasificación**: es la tarea más frecuente en minería de datos. Cada registro de la base de datos (llamado *instancia*) tiene asignado una clase, de entre un número finito de clases, en uno de sus atributos, que suele ser de tipo discreto. El objetivo de la tarea es predecir la clase de una instancia en función de los valores del resto de sus atributos.

  Por ejemplo, un problema de clasificación podría ser aquel en el que cada instancia es una fotografía del rostro de una persona y el objetivo del modelo es identificar si la persona es un hombre o una mujer.

- **Regresión**: la regresión es un problema con un planteamiento similar al problema de clasificación, con la diferencia de que la variable a predecir es un valor continuo en lugar de un valor discreto. Concretamente, la tarea consiste en "aprender una función real que asigna a cada instancia un valor real".

❖ **Tareas no supervisadas**: el sistema de aprendizaje no recibe etiquetas sobre los datos y debe aprender por sí mismo la estructura de los datos de entrada.

- **Agrupamiento** (*clustering)*: el agrupamiento es una tarea de descripción de datos que consiste en obtener grupos en los datos. A diferencia de la clasificación, los datos no tienen una clase asignada, sino que se buscan grupos de instancias similares entre sí. En concreto, se buscan grupos de tal forma que se maximice la similitud entre elementos de un mismo grupo y se minimice la similitud entre elementos de distintos grupos.
- **Correlaciones**: se trata de otra tarea descriptiva para analizar el grado de similitud entre los valores de dos variables numéricas. Se suele utilizar el coeficiente de correlación *r*, que toma valores entre -1 y 1. El coeficiente tomará el valor 1 cuando las variables estén perfectamente relacionadas positivamente, 0 cuando no haya correlación y -1 cuando estén perfectamente relacionadas negativamente.
- **Reglas de asociación**: son otro tipo de tarea descriptiva que busca relaciones entre atributos categóricos. Son muy utilizadas en análisis de cesta de la compra para encontrar productos que se compran juntos frecuentemente, sin que haya una relación evidente entre ellos; las

reglas de asociación suelen tener la forma “Si el atributo X toma el valor x entonces el atributo Y toma el valor y”.

- **Reglas de asociación secuenciales**: son un caso especial de reglas de asociación en el que las relaciones entre los datos se basan en el tiempo. Darían lugar a reglas de la forma “Si se compra el producto X, entonces en un plazo de Y días, se comprará el producto Z”.

- **Tareas por refuerzo:** se proporciona al sistema unas entradas, en forma de recompensas o castigos, como realimentación de las acciones que el sistema lleva a cabo en un entorno dinámico.

2.1.3.2. Técnicas

En minería de datos no existe un método universal que nos permita resolver todos los problemas, sino que existen múltiples paradigmas, que incluyen distintos algoritmos, que nos permiten resolver diferentes problemas en función de cuál sea la tarea a resolver, el dominio de la aplicación, las características de los datos, etc. Algunas de las técnicas más utilizadas en minería de datos son:

- **Técnicas estadísticas**: muchas técnicas de minería de datos están basadas en conceptos estadísticos útiles para resolver distintas tareas. Por ejemplo, el método de la regresión lineal es muy utilizado en la tarea de regresión o los análisis de discriminantes lineal de Fisher [2-7.8] utilizados tanto para clasificación como para agrupamiento.
- **Métodos bayesianos**: estos métodos buscan, para una instancia dada sin clasificar, cuál es la probabilidad de que se asigne a cada una de las clases posibles y seleccionar aquella de mayor probabilidad. El método más conocido es el *Naïve Bayes* [2-10], que asume “ingenuamente” la independencia de los atributos.
- **Métodos basados en núcleo**: el objetivo de estos métodos es encontrar un discriminante o frontera entre elementos de diferentes clases. Uno de los métodos basados en núcleo más conocidos y utilizados es el de la máquina de vectores soporte (*Support Vector Machine, SVM*) [2-14], que busca una frontera lineal que maximice la distancia a los ejemplos de los diferentes grupos o clases.
- **Árboles de decisión**: estos métodos crean una serie de decisiones organizadas jerárquicamente en forma de árbol. Es una técnica muy utilizada

en clasificación (árboles de clasificación), en agrupamiento y en regresión (árboles de regresión). Algunos autores consideran los árboles como una forma de aprendizaje de reglas, ya que cada camino desde la raíz del árbol hasta un nodo hoja puede considerarse como una regla.

Uno de los algoritmos de árboles de decisión más populares (sino el más popular) es C4.5 [12]. Este algoritmo divide el conjunto de datos de entrenamiento en cada nodo según un atributo, elegido mediante **ganancia de información**. Este proceso continúa recursivamente hasta llegar a unos pocos casos base, en los que el algoritmo crea los nodos hoja, que indican la clase asignada.

- **Inducción de reglas**: son métodos que pretenden encontrar un conjunto de reglas de la forma:

  $\texttt{SI cond}_1\ \texttt{Y cond}_2\ \texttt{Y…}\ \texttt{Y cond}_n\ \texttt{ENTONCES predicción}$

  El antecedente de la regla (la parte `SI`) está formado por una conjunción de condiciones sobre los atributos de la instancia. El consecuente (la parte `ENTONCES`) contiene una predicción sobre el valor de la variable objetivo.

  Aunque como se ha mencionado antes, los métodos de árboles de decisión pueden considerarse métodos de aprendizaje por reglas, no todos los métodos de aprendizaje por reglas generan reglas que puedan organizarse en forma de árbol.

- **Redes neuronales artificiales**: son un paradigma de aprendizaje que permiten modelar problemas muy complejos en los que puede haber relaciones no lineales entre las variables. Se pueden utilizar para clasificación, regresión y agrupamiento. Una restricción importante de las redes neuronales es que solamente pueden trabajar con datos numéricos, por lo que las variables nominales tienen que *numerizarse* primero.

  Puesto que uno de los objetivos principales de este proyecto es la familiarización y utilización de redes neuronales artificiales, estas se describen en detalle en el punto 2.2.

- **Métodos basados en instancias**: son métodos que, ante una nueva instancia de clase desconocida, predicen la clase en función de su similitud con el resto de elementos de la base de datos. A menudo se les llama métodos

perezosos ya que no crean un modelo utilizando la base de datos, sino que retrasan todo el trabajo al momento en el que aparece una nueva instancia a clasificar. Uno de los métodos más conocidos es el “k vecinos más próximos” (*k nearest neighbors, kNN*) [13], que asigna a la instancia la clase mayoritaria de los k elementos más cercanos o similares a esta instancia.

- **Algoritmos evolutivos**: son métodos de búsqueda en el espacio de todas las soluciones posibles, inspirados en la evolución biológica. Se pueden utilizar en agrupamiento, clasificación, reglas de asociación y selección de atributos. También son muy empleados para abordar problemas de optimización en general.

2.1.3.3. Construcción del modelo

En esta fase se “entrena” el modelo con los datos disponibles. Es habitual en este punto volver a fases anteriores para preparar los datos de forma adecuada según el método elegido o simplemente para probar otros modelos hasta encontrar el más adecuado a la tarea a resolver con los datos disponibles.

2.1.4. Evaluación e interpretación

En esta fase se tiene como objetivo la medida de la calidad de los patrones encontrados por el algoritmo de minería de datos. Dichos patrones deberían ser precisos, comprensibles e interesantes. Estas propiedades son habitualmente opuestas unas a otras, por lo que hemos de seleccionar y medir el modelo que nos optimice una de ellas, en detrimento de las otras.

Para entrenar y probar un modelo se parten los datos en tres subconjuntos: el conjunto de entrenamiento (*training set*), el conjunto de validación (*validation set*)[1] y el conjunto de test o prueba (*test set*). Esta división del conjunto de datos garantiza la independencia de la medida de precisión del modelo, evitando que se sobreestime.

La función de cada uno de estos subconjuntos es la siguiente:

- **Subconjunto de entrenamiento**: se utiliza para construir (entrenar) el modelo que extrae los patrones de los datos.

[1] En algunas de las referencias bibliográficas no se considera el conjunto de validación como un conjunto independiente, sino como una división del conjunto de entrenamiento. En otras referencias se considera el conjunto de validación como recomendable pero no obligatorio, dependiendo de la tarea a resolver y del algoritmo a utilizar, especialmente si la cantidad de datos disponibles no es muy elevada.

- **Subconjunto de validación**: se utiliza para ajustar los distintos parámetros del modelo entrenado.
- **Subconjunto de test**: se utiliza para medir la bondad o precisión del modelo ya entrenado y ajustado.

Un error muy común es utilizar el conjunto de test para realizar el ajuste de los parámetros del modelo. Esto provoca que la medida de la precisión del modelo no sea real, sino una sobreestimación que no es independiente de los datos. La norma es que el conjunto de test sólo se utiliza una vez, al final del proceso, para medir la precisión del modelo ya entrenado y ajustado.

2.1.4.1. Métodos de evaluación

La precisión de un modelo se obtiene dividiendo el número de instancias clasificadas correctamente entre el número total de instancias a clasificar. Esta precisión es una buena estimación de cómo se comportará el modelo ante futuras instancias desconocidas, pero no garantiza que el modelo sea correcto.

Existen varios métodos de evaluación:

- **Validación simple**: se reserva un porcentaje de la base de datos, de forma aleatoria, como conjunto de prueba. Para mejorar los resultados de la evaluación se puede hacer que el conjunto de entrenamiento y el de prueba tengan la misma distribución de datos de cada clase. Esto se conoce como **estratificación**.
- **Validación cruzada con k pliegues** (*K-fold cross validation*): en este método se divide el conjunto de datos en *k* subconjuntos de igual tamaño, construyendo el modelo con *k-1* de estos subconjuntos y utilizando el restante como conjunto de prueba. Este proceso se repite *k* veces, utilizando cada vez un subconjunto diferente como conjunto de prueba. Finalmente se construye un modelo con todos los datos y se obtienen sus ratios de error promediando las ratios de los modelos de la validación cruzada.

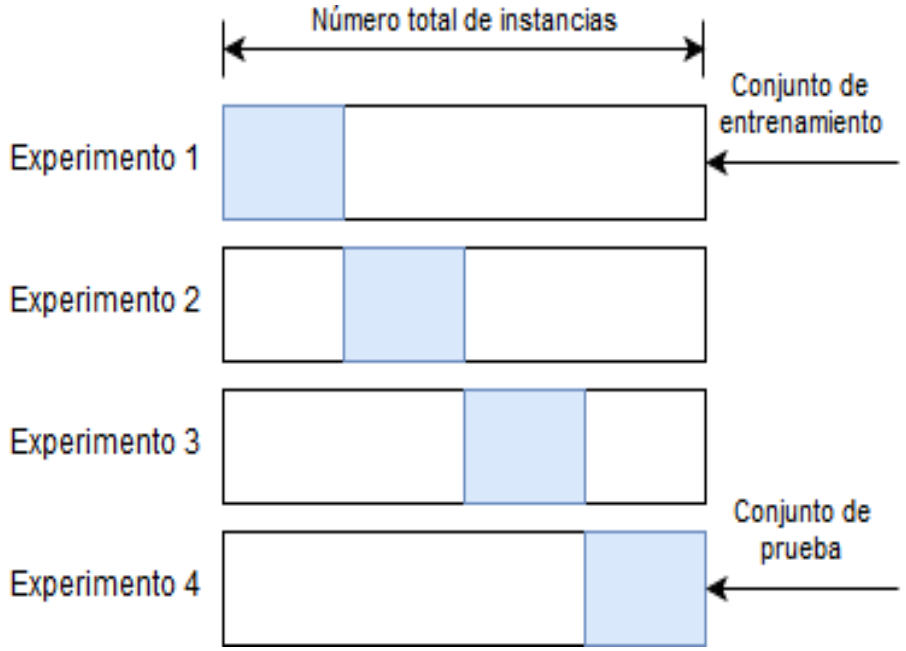


Ilustración 4 Validación cruzada con k=4 pliegues. (Fuente: Elaboración propia).

- **Leave-One_out cross validation**: este método es el caso extremo de la validación cruzada, en el que *k* es el tamaño de la base de datos (*k=n*). De esta forma se construyen *n* modelos utilizando *n-1* instancias para entrenamiento y una sola instancia para prueba.

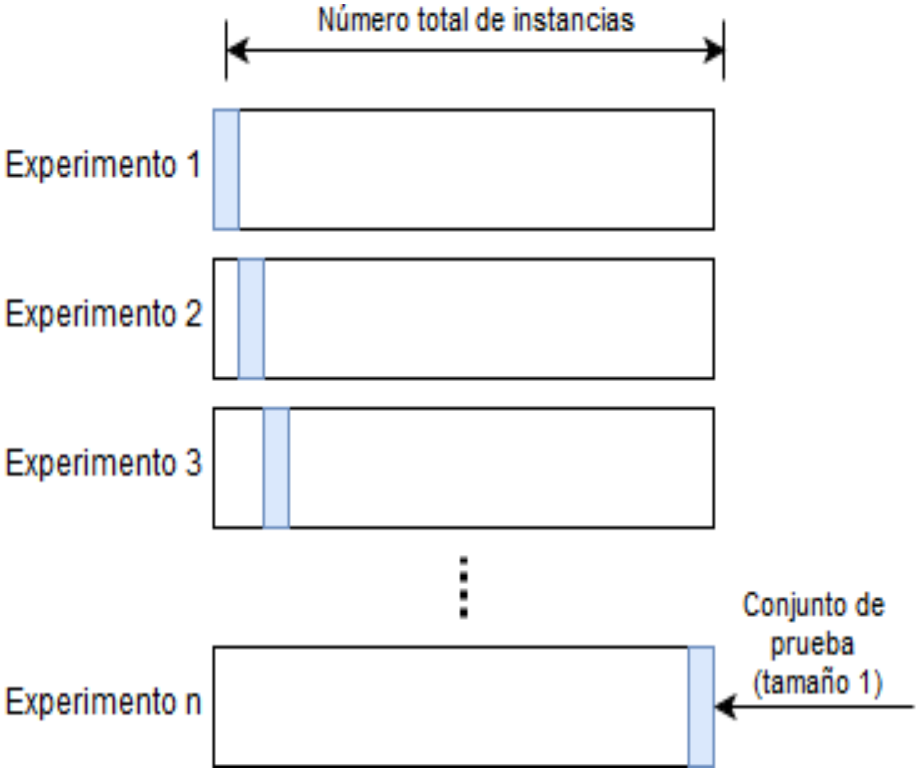


Ilustración 5 Validación leaving one out. (Fuente: Elaboración propia).

- **Muestreo aleatorio**: en este método se construyen varios modelos, extrayendo en cada iteración un conjunto de prueba diferente de forma aleatoria.

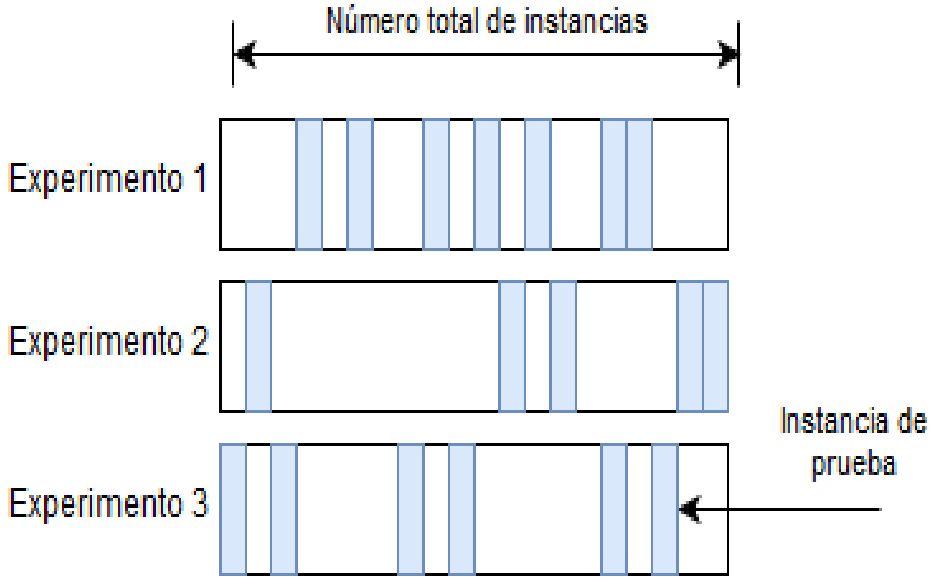


Ilustración 6 Validación con muestreo aleatorio. (Fuente: Elaboración propia).

- **Bootstrapping**: en este método también se realizan varios experimentos, seleccionando aleatoriamente los datos para entrenamiento y para test, con la diferencia de que el muestreo se realiza con reemplazo, permitiéndose seleccionar la misma instancia en varias ocasiones.

2.1.4.2. Medidas de evaluación

Existen diferentes medidas para la evaluación de modelos, en función de la tarea de minería de datos a resolver.

- **Clasificación**: la medida más habitual es la **precisión predictiva** obtenida dividiendo el número de instancias del conjunto de prueba clasificadas correctamente entre el número total de instancias en el conjunto de prueba. En el apartado 2.1.4.3 se describen en detalle esta y otras medidas que se pueden utilizar en la tarea de clasificación.
- **Reglas de asociación**: habitualmente se evalúa cada una de las reglas por separado, utilizando dos conceptos:
  - **Cobertura** (o soporte): número de instancias a las se aplica la regla y se predice correctamente.

- **Confianza**: proporción de instancias que la regla predice correctamente. Se calcula dividiendo el soporte por el número de instancias a las que se puede aplicar la regla.

❖ **Regresión**: puesto que en esta tarea la salida es un valor real, la medida más usada es el **error cuadrático medio** del valor predicho con respecto al valor de prueba o validación.

❖ **Agrupamiento**: en esta tarea las medidas de evaluación dependen del método de minería de datos utilizado, aunque suelen ser medidas en función de la cohesión de los elementos de cada grupo y de la separación entre grupos.

2.1.4.3. Interpretación y contextualización

Habitualmente no basta con estas medidas de evaluación, sino que se ha de tener en cuenta el contexto donde se utilizará el modelo. Las medidas como la precisión en clasificación no tienen en cuenta problemas como el desbalanceo de clases, es decir, tener muchas instancias de unas clases y pocas instancias de otras clases. Esta situación es muy habitual, por ejemplo, en problemas de fraudes bancarios, donde el número de casos en los que no hay fraude es muy superior a los casos en los que sí hay fraude. Suponiendo que tuviéramos un 95% de instancias de "no fraude" y un 5% de instancias de "fraude", un modelo que clasificara siempre como "no fraude" tendría una precisión del 95%, pero no sería útil ya que no serviría para detectar el problema que nos interesa, que es la detección de fraudes.

Otra forma de medir los errores en problemas de clasificación es utilizando la **matriz de confusión**. En esta matriz se muestra la cantidad de instancias predichas para cada clase, así como la cantidad de instancias de cada clase. Por ejemplo, para una clasificación binaria de análisis de fraudes (los datos no son reales, sino que son para ilustrar este ejemplo) podemos obtener la siguiente matriz de confusión:

	Predicción: fraude	Predicción: no fraude
Etiqueta: fraude	2	3
Etiqueta: no fraude	4	991

Tabla 1 Matriz de confusión con datos no reales. (Fuente: elaboración propia)

Es común en la literatura referirse a estas cuatro celdas de la matriz de confusión con nombres derivados de una clasificación en la que las clases son “acierto” y “fallo” o “positivos” y “negativos”. Suponiendo que la clase “fraude” de nuestro ejemplo sean aciertos y la clase “no fraude” sean fallos tendríamos:

- **Verdaderos positivos** (**VP**): aquellas instancias que pertenecen a la clase “fraude” y que se han etiquetado correctamente. En nuestro ejemplo serían 2 instancias.
- **Falsos positivos** (**FP**): aquellas instancias que se han etiquetado como clase “fraude”, siendo en realidad de la clase “no fraude”. En nuestro ejemplo, 4 instancias.
- **Falsos negativos** (**FN**): instancias etiquetadas como clase “no fraude”, perteneciendo a la clase “fraude”. En el ejemplo, 3 instancias.
- **Verdaderos negativos** (**VN**): aquellas instancias que pertenecen a la clase “no fraude” y que se han etiquetado correctamente. En el ejemplo, 991 instancias.

A partir de esta matriz de confusión se pueden obtener una serie de métricas para la evaluación de la clasificación:

- **Precisión predictiva** (*accuracy*): tal y como se ha mencionado antes, se calcula dividiendo el número de instancias clasificadas correctamente entre el número total de instancias.

$$Accuracy = \frac{VP + VN}{VP + VN + FP + FN}$$

$$Accuracy = \frac{2 + 991}{2 + 991 + 4 + 3} = 0.993$$

- ***Precision***: esta medida calcula qué porcentaje de las instancias clasificadas como aciertos se han clasificado correctamente. Se calcula dividiendo los verdaderos positivos entre todas las instancias etiquetadas como positivas (VP + FP).

$$Precision = \frac{VP}{VP + FP}$$

$$Precision = \frac{2}{2 + 4} = 0.333$$

- ***Exhaustividad*** *(Recall)*: esta medida calcula el porcentaje de instancias positivas clasificadas correctamente. Se obtiene dividiendo los verdaderos positivos entre todas las instancias que son positivas.

$$Recall = \frac{VP}{VP + FN}$$

$$Recall = \frac{2}{2 + 3} = 0.4$$

- ***F1-Score***: es la media harmónica entre las medidas *Precision* y *Recall*.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

$$F1 = \frac{2 * 0.333 * 0.4}{0.333 + 0.4} = 0.363$$

Como se puede ver en los cálculos realizados con los datos del ejemplo, este modelo obtendría un *accuracy* excelente (99.3% de acierto), no obstante, en el resto de medidas, más representativas del acierto en el problema que queremos resolver, los resultados obtenidos son bastante malos. Por lo tanto, se puede decir que este sistema no resuelve correctamente el problema para el que ha sido creado.

2.1.5. Difusión, uso y monitorización

Una vez construido y validado el modelo, este puede utilizarse para varias finalidades:

- Para que un experto o un analista recomiende acciones apoyándose en el modelo y en el conocimiento extraído por este.
- Para aplicar el modelo a diferentes conjuntos de datos.
- Para incorporarse a otras aplicaciones de forma automática (filtros de spam) o semiautomática (sistemas de asistencia).

En cualquiera de los casos es necesario que el modelo se distribuya a los nuevos posibles usuarios. No menos importante es la supervisión del modelo, es decir, medir cómo este evoluciona según su uso, nuevos datos, etc. Pasado un tiempo es probable que el modelo tenga que ser reevaluado, reentrenado o reconstruido completamente.

3. Deep Learning

En este capítulo se exponen los fundamentos de las redes neuronales artificiales, las redes neuronales convolucionales y los elementos que las componen. También se presentan algunas de las arquitecturas y *frameworks* o entornos de trabajo con redes neuronales convolucionales más populares en la actualidad.

3.1. Redes neuronales artificiales

Las redes neuronales artificiales (*Artificial Neural Network, ANN*) son sistemas de aprendizaje inspirados en las redes neuronales biológicas que forman los cerebros de los animales.

Una ANN es una red de elementos simples (llamado neuronas artificiales), que reciben una entrada, cambian su estado interno (activación) según esa entrada y producen una salida en función de esa entrada y de su activación.

La red se forma conectando las salidas de unas neuronas a las entradas de otras, formando un grafo dirigido ponderado. El proceso de aprendizaje modifica los valores de los pesos del grafo, así como las salidas de las funciones de activación de las neuronas para lograr la mejor clasificación posible de los datos.

Estas neuronas se distribuyen en capas, de forma que las neuronas de la capa *n* toman como entradas las salidas de las neuronas de la capa *n–1.* A su vez, las salidas de la capa *n* sirven como entradas a las neuronas de la *n+1*. La primera capa de la red se conoce como **capa de entrada** (*input layer*), la última capa se conoce como **capa de salida** (*output layer*) y el resto de capas intermedias entre la capa de entrada y la capa de salida se conocen como **capas ocultas** (*hidden layers*).

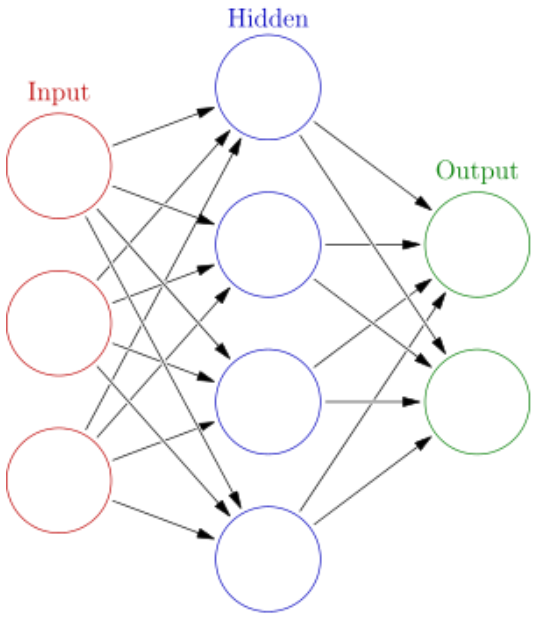


Ilustración 7 Estructura básica de una red neuronal. (Fuente: Wikipedia)

En general, se denomina **arquitectura de una red neuronal artificial** a la configuración de las capas (número de capas y número de neuronas por capa) de la red junto a la **función de activación** de las neuronas. Habitualmente, todas las neuronas de una misma capa tienen la misma función de activación.

Existen múltiples modificaciones de la arquitectura de las redes neuronales, según el tipo de problema de minería de datos a resolver, según el tipo de datos disponibles, etc. Algunas de las arquitecturas más populares son:

- **Perceptrón** (o perceptrón simple) [14]: el perceptrón está compuesto por una sola neurona, siendo la arquitectura de red neuronal más simple que existe. Se utiliza como método de clasificación binaria que, a partir de un vector de valores numéricos, busca una separación lineal entre las dos posibles clases de salida.

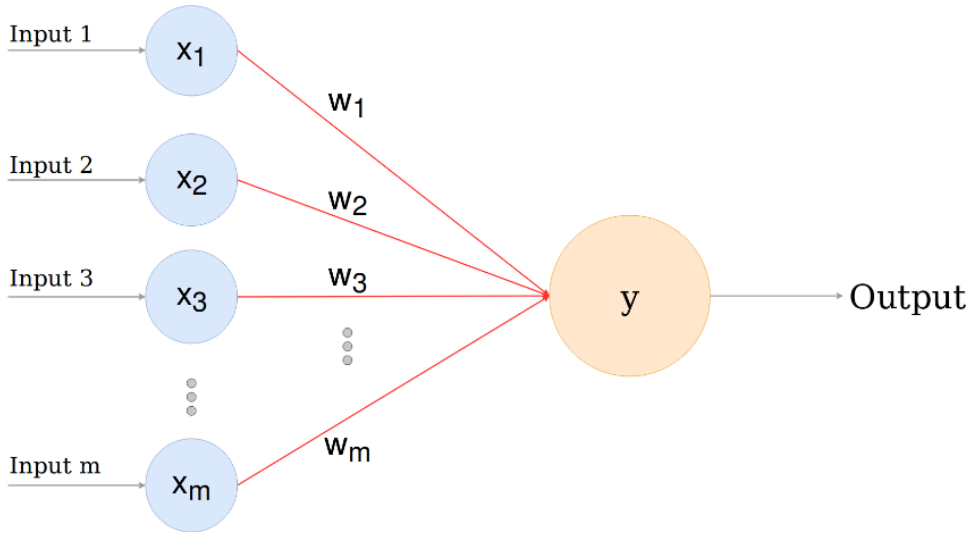


Ilustración 8 Ejemplo de arquitectura Perceptrón. (Fuente: Towards Data Science)

- **Perceptrón multicapa** (*Multilayer Perceptron*): esta arquitectura es la más conocida cuando se habla de ANNs (la Ilustración 7 es un ejemplo de perceptrón multicapa). Esta arquitectura tiene capacidad para resolver problemas que no son linealmente separables, siendo esta la limitación más importante del perceptrón simple.
- **Redes neuronales profundas** (*Deep Neural Networks*): son redes neuronales con gran cantidad de capas ocultas y de neuronas en cada capa. Estas redes son capaces de modelar relaciones no lineales muy complejas, aunque son propensas al sobreaprendizaje y necesitan de la consideración de muchos parámetros (número de capas, número de neuronas en cada capa, ratio de aprendizaje, etc.) además de una elevada capacidad de computo.
- **Redes neuronales convolucionales** (*Convolutional Neural Networks*, *CNNs*): son una clase de redes neuronales profundas muy utilizadas en análisis de imágenes. Para la clasificación de imágenes requieren de muy poco preprocesamiento, comparadas con otros sistemas de clasificación.

  Están formadas por una capa de entrada, una capa de salida y múltiples capas ocultas que pueden ser: capas convolucionales, capas de reducción de muestreo (*pooling*), capas completamente conectadas o capas de normalización.

  Puesto que este tipo de redes son el objeto de estudio de este proyecto, se verán más detalladamente más adelante.

- **Redes neuronales recurrentes** (*Recurrent Neural Networks, RNNs*): son un tipo de redes neuronales artificiales donde las conexiones entre neuronas forman un grafo dirigido a lo largo de una secuencia, lo que les permite simular una "memoria" a la hora de procesar secuencias de datos. Se suelen utilizar para procesamiento de texto manuscrito o del habla.

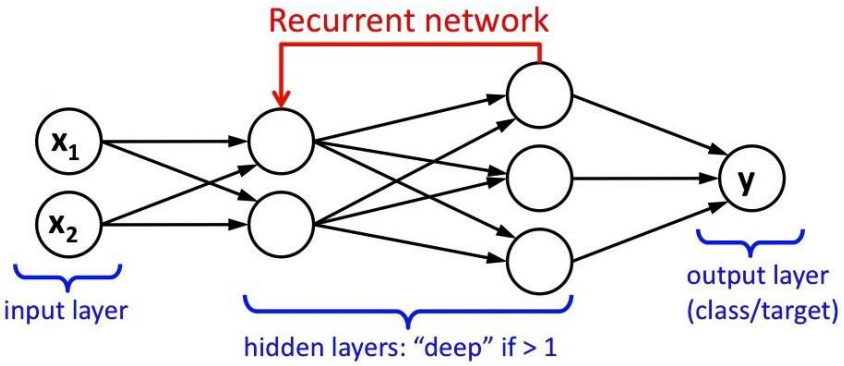


Ilustración 9 Ejemplo red neuronal recurrente. (Fuente: medium.com)

3.2. Redes neuronales convolucionales

En los últimos años las CNNs han mostrado los mejores resultados en todo tipo de procesamiento de imágenes, en particular en su clasificación.

El funcionamiento de una CNN se puede descomponer en dos etapas. En la primera, utilizando capas convolucionales y capas de reducción de muestreo (*pooling layers*) la red extrae características de las imágenes. Las primeras capas de esta etapa extraen características (*feature extraction)* simples, como líneas, que se van agregando en capas sucesivas para extraer formas más complejas, desde figuras geométricas simples hasta vehículos, rostros o personas.

En la segunda etapa se utilizan estas características en un modelo que distinga entre las diferentes clases que queremos clasificar.

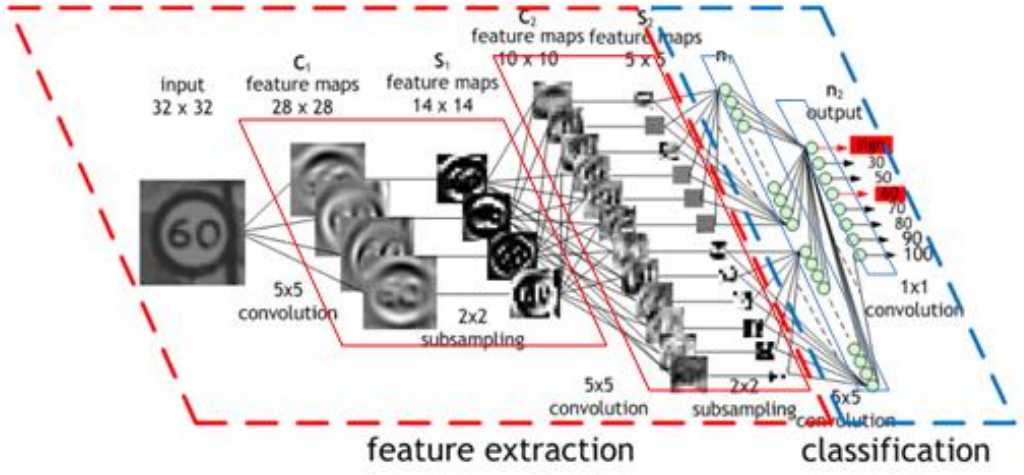


Ilustración 10 Ejemplo de red convolucional para clasificación de señales de tráfico. (Fuente: NVIDIA)

Además de las capas que componen estas dos etapas, la red posee una capa de entrada, donde se introduce la imagen, y una capa se salida, que produce el resultado de la clasificación.

A continuación, se describen los diferentes elementos que componen una CNN.

3.2.1. Elementos de una CNN

Como ya se ha mencionado anteriormente, una CNN está formada por una capa de entrada, un grupo de capas de extracción de características (compuestas de capas convolucionales, capas de activación y capas de reducción de muestreo)[2], seguidas de una serie de capas completamente conectadas y una última capa se salida. También se consideran elementos de la red las funciones de activación de las neuronas, así como el resto de *hiperparámetros* configurables de la red.

Distinguimos entre los términos **parámetros** como aquellos valores que la red intenta optimizar en su proceso de aprendizaje e **hiperparámetros** como aquellos valores configurables de la red que son decididos por el diseñador de la red.

3.2.1.1. Capas convolucionales (*Convolutional layers*)

Este tipo de capas son en las que se sustentan las CNNs (de hecho, las redes convolucionales reciben su nombre de este tipo de capas). En ellas se aplica la operación de **convolución** sobre las imágenes generando la activación de ciertas

[2] El diseño más común en esta etapa es el de varias repeticiones de la siguiente estructura: una o varias capas convolucionales, una capa de activación y una capa de muestreo.

neuronas. Estas convoluciones, filtros o *kernels* extraen información de un grupo reducido de píxeles de la imagen (habitualmente cuadrados de 3x3 o 5x5). La operación se aplica sobre toda la imagen, comenzando en una esquina de esta, y trasladando el filtro a lo largo y ancho de toda la imagen.

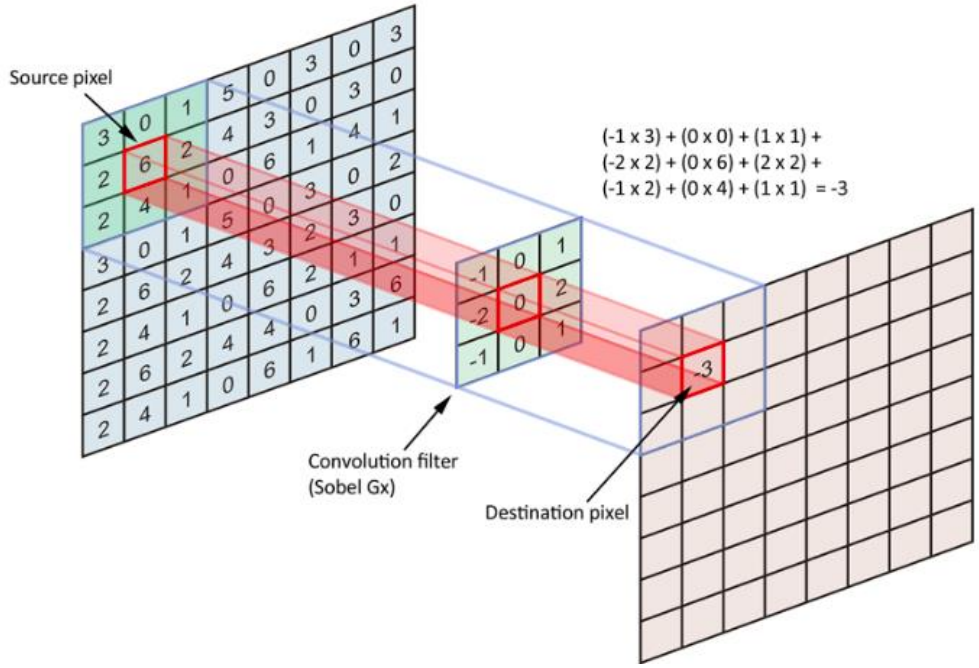


Ilustración 11 Operación de convolución. (Fuente: FreeCodeCamp)

Estas capas convolucionales pueden configurarse de acuerdo a los siguientes hiperparámetros:

- **Tamaño del filtro**: el tamaño del filtro indica el campo de visión de la capa, es decir, el número de píxeles a los que se aplicará el filtro. Habitualmente se utilizan filtros cuadrados de tamaño impar (3x3, 5x5 o 9x9, por ejemplo). También es importante mencionar que el filtro ha de tener en cuenta el número de canales de la imagen (3 en imágenes a color RGB, 1 en imágenes en blanco y negro), aunque esto se suele obviar. Por tanto, un filtro de tamaño 5x5 para una imagen a color será, en realidad, de tamaño 5x5x3.

  Algunos investigadores [28] sugieren que es preferible utilizar varios filtros sucesivos de pequeño tamaño a utilizar un filtro de gran tamaño.
- **Número de filtros**: en una misma etapa de convolución se pueden aplicar no solo uno sino varios filtros del mismo tamaño a la imagen, aumentando el número de características que se pueden extraer en un solo paso.
- **Pasos** (*strides*): antes se ha mencionado que el filtro se aplica a la imagen comenzando en una esquina y se va trasladando a lo largo y ancho de la

imagen. Esta traslación puede hacerse moviendo el filtro un solo píxel en una de las direcciones (*stride=1*) o, por ejemplo, dos píxeles (*stride=2*).

- **Relleno** (*Padding)*: la operación de convolución, por su naturaleza, reduce el tamaño de la imagen, limitando el número de veces que podemos aplicar la operación. Otra desventaja es que los píxeles de los bordes y esquinas de la imagen se utilizan menos veces que los píxeles interiores, "desperdiciando" parte de la información de la imagen. La solución a este problema es añadir un borde alrededor de la imagen original, con 0 como valores.

  Los valores más frecuentes para este hiperparámetro son ***Valid***, que indica que no se añade un borde y ***Same***, que añade un borde de forma que el resultado de la convolución tenga el mismo tamaño que la imagen original.

3.2.1.2. Capas de reducción de muestreo o submuestreo (*Pooling layers*)

Estas capas se utilizan combinadas con las capas convolucionales para reducir el tamaño de las representaciones, para acelerar la computación y para hacer que algunas de las características extraídas sean más robustas.

El tipo de submuestreo más habitual es el *Max Pooling* que aplica filtros a la imagen, extrayendo el mayor valor dentro del filtro. También existe el *Average Pooling*, que extrae la media de los valores dentro del filtro, aunque es mucho menos utilizado que el *Max Pooling*.

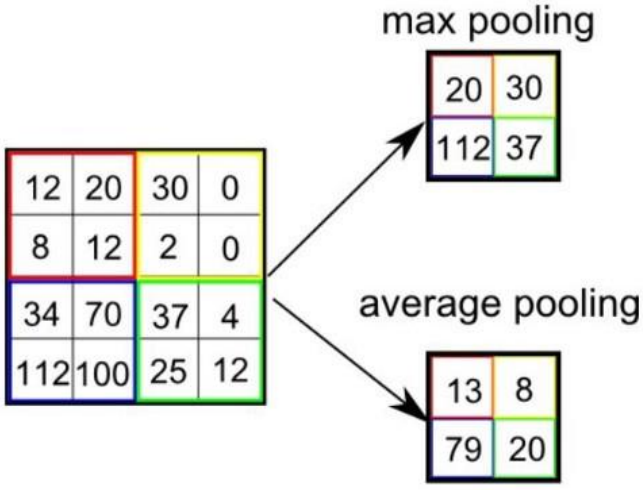


Ilustración 12 Tipos de submuestreo, con tamaño 2x2 y paso de 2 píxeles. (Fuente: Towards Data Science)

Los hiperparámetros de estas capas son muy similares a los de las capas convolucionales:

- Tamaño del filtro
- Relleno (*padding*)
- Pasos (*stride*)

Además de reducir el tamaño de la representación interna de la imagen, estas capas presentan la ventaja de no tener parámetros internos que el algoritmo de aprendizaje tenga que optimizar.

3.2.1.3. Capas completamente conectadas (*Fully connected layers*)

Estas capas se utilizan como capas finales del modelo, situadas antes de la capa de salida. Su nombre se debe a que la salida de una neurona en la capa *n* está conectada con las entradas de todas las neuronas de la capa *n+1*.

Este tipo de capas son habituales en todos los tipos de redes neuronales artificiales, no solamente en la CNNs, y es en ellas donde se produce la diferenciación o clasificación de las imágenes en las posibles clases.

Los hiperparámetros de este tipo de capas son:

- Número de capas.
- Número de neuronas en cada capa.
- Función de activación de las neuronas de una capa.

La capa interna de la Ilustración 7 es un ejemplo de capa completamente conectada.

3.2.1.4. Capa de salida

En esta capa es donde se toma la decisión de la clase a la que pertenece la imagen, utilizando como entrada las activaciones de las capas completamente conectadas. Esta capa tiene tantas neuronas como clases tenga nuestro problema de clasificación.

3.2.1.5. Funciones de activación

Las funciones de activación se utilizan para transformar la salida de un nodo, decidiendo la influencia que tendrá en el proceso de entrenamiento. Las funciones utilizadas más frecuentemente son:

- **Sigmoide**: función que toma cualquier valor real como entrada y produce un resultado entre 0 y 1. Su curva de salida tiene la característica forma de letra "S". Se utiliza, entre otros, en la capa de salida cuando tenemos un problema de clasificación binaria (dos clases).

$$f(t) = \frac{1}{1 + e^{-t}}$$

- **Softmax**: la función exponencial normalizada (función *Softmax*) calcula la distribución de probabilidad de un evento sobre *n* posibles eventos. Es decir, calcula la probabilidad de cada clase objetivo, sobre todas las posibles clases objetivos. El rango de las salidas está en el intervalo [0, 1], cumpliéndose, además, que la suma de todas las salidas es igual a 1.

  Se utiliza habitualmente en la capa se salida para clasificación multiclase, ya que nos devuelve la probabilidad de que la instancia pertenezca a cada una de las clases.

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{k} e^{x_j}}, i = 0, 1, 2, \dots k$$

- **TanH**: la función tangente hiperbólica produce una salida similar a la función sigmoide, pero con un rango de salida en el intervalo [-1, 1].

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- **RELU**: la función rectificadora lineal (*REctified Linear Unit, RELU*), también conocida como la función rampa, es una de las más utilizadas actualmente en redes convolucionales.

$$f(x) = \max(0, x)$$

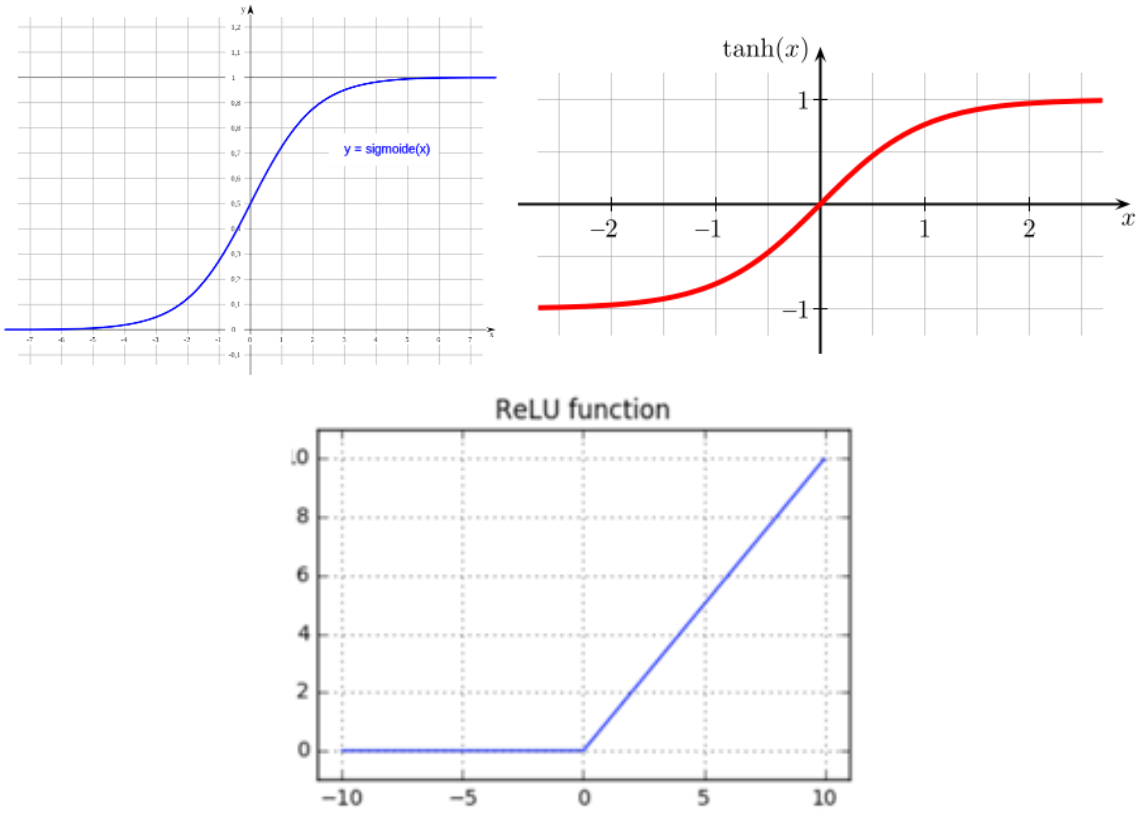


Ilustración 13 Funciones de activación. Sigmoide (arriba izqda.), TanH (arriba dcha.) y RELU (abajo). (Fuentes: Wikipedia (1 y 2), medium.com (3))

3.2.1.6. Otros hiperparámetros de la red

Aparte de los hiperparámetros de cada capa, existen otros propios de la red en si misma:

- **Tamaño del lote** (*batch size*): este parámetro define el tamaño de los subconjuntos en los que se dividirá el conjunto de entrenamiento a la hora se ser procesado por la red neuronal. Este parámetro es muy importante, pues los parámetros internos de la red se actualizarán al final de cada uno de estos lotes. Es habitual utilizar potencias de dos (16, 32, 64, etc.) o divisores exactos del número de instancias de entrenamiento. Un tamaño de lote alto hará que se entrene la red más rápidamente, pues los parámetros se actualizarán menos veces, pero necesitará más memoria para trabajar. Un tamaño pequeño, por el contrario, necesitará mayor tiempo para entrenar la red, pero necesitará de menos memoria.
- **Número de etapas** (*epochs*): número de veces que el algoritmo de aprendizaje procesará completamente el conjunto de entrenamiento. Al final de cada uno de estos *epochs* el algoritmo utilizará el conjunto de validación para comprobar

y ajustar los parámetros internos de la red, en función del error cometido al clasificar estos datos.

Utilizar un número reducido de *epochs* puede hacer que la red no tenga ocasión de hacer un ajuste adecuado de los parámetros, mientras que uno demasiado elevado puede llevar a un sobreajuste o sobreaprendizaje (explicado en más detalle en el apartado 3.2.2).

- **Dropout**: es una técnica desarrollada para reducir el sobreajuste en el entrenamiento de una red. Consiste en ignorar aleatoriamente en cada *epoch,* durante el proceso de entrenamiento, un porcentaje de los nodos, tanto en capas ocultas como en capas visibles. Además de reducir el sobreajuste, permite un ahorro en la cantidad de computación necesaria.

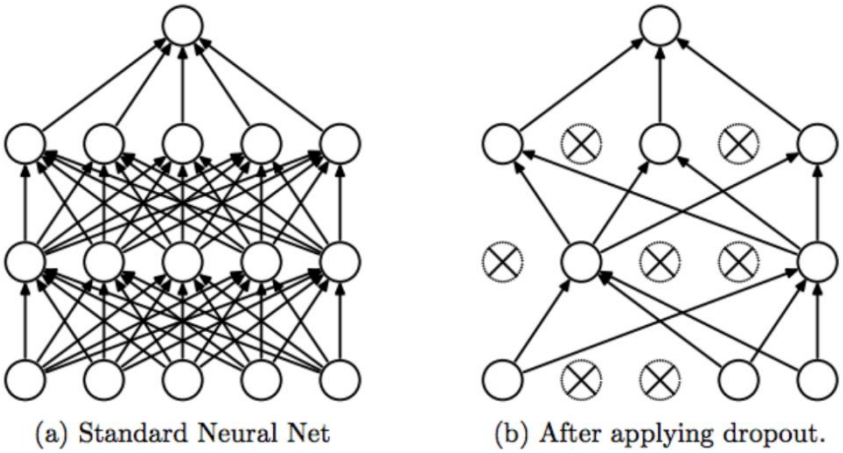


Ilustración 14 Ejemplo de uso del Dropout. (Fuente: [5])

Generalmente, el hiperparámetro de *Dropout* se aplica en las capas completamente conectadas e indica el porcentaje de neuronas a desactivar en esa capa en cada *epoch*.

- **Aumento o aumentador de datos**: se trata de otra técnica utilizada para reducir el sobreaprendizaje y tener una mayor variedad de ejemplos de imágenes de las que aprender. Esta técnica consiste en aplicar algunas transformaciones aleatorias a las imágenes de entrenamiento, de forma aleatoria en cada *epoch*, para obtener diferentes muestras de la misma imagen. Algunas de las transformaciones más habituales son la rotación, la traslación, la ampliación o reducción (*zoom*), alteración de los colores, del brillo o de la saturación y la inversión sobre el eje vertical.

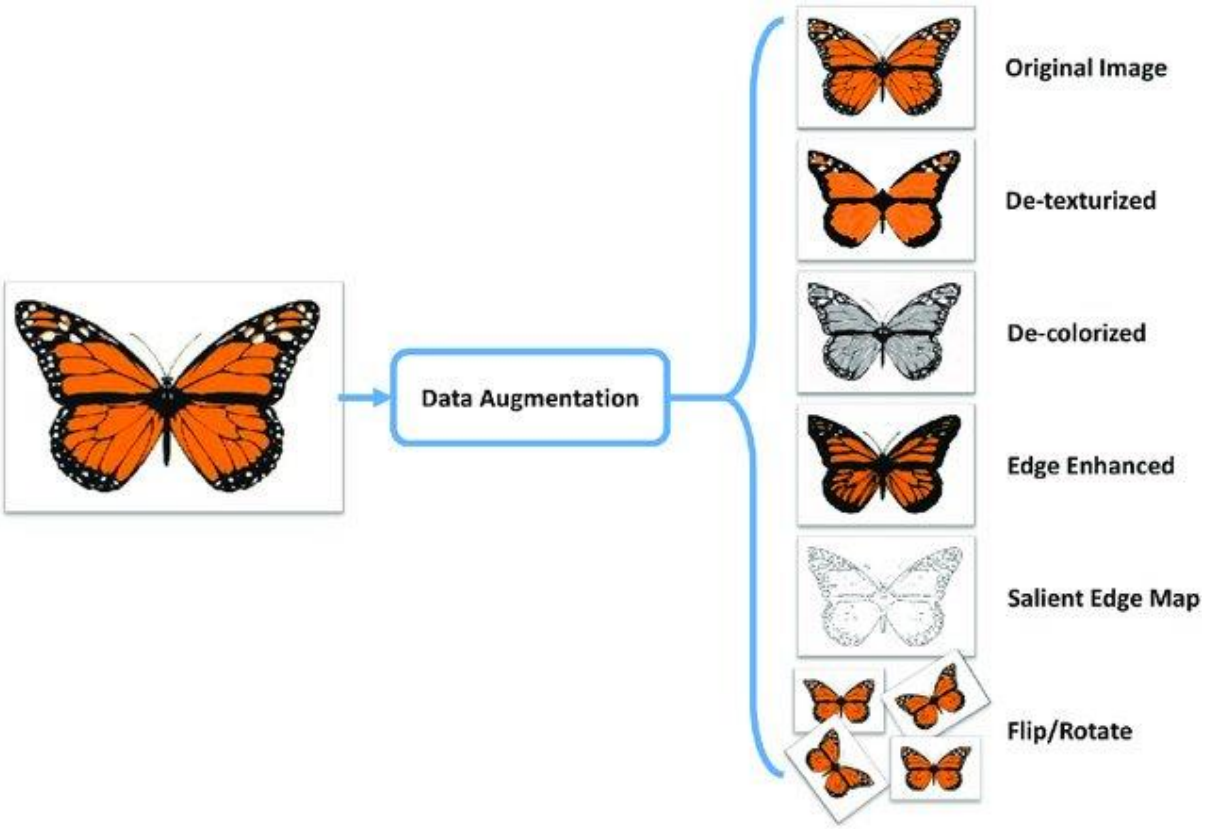


Ilustración 15 Ejemplos de aumento de datos sobre la imagen de una mariposa. (Fuente: [5])

3.2.2. El problema del sobreaprendizaje

Un problema habitual que se presenta en el entrenamiento de redes neuronales profundas (se presenta en todos los modelos de aprendizaje automático, pero especialmente en las redes profundas) es el conocido como **sobreaprendizaje o sobreajuste** (*overfitting* en inglés). Cuando tenemos demasiados parámetros, el sistema se ajusta muy bien al conjunto de aprendizaje, pero falla al generalizar con nuevos ejemplos.

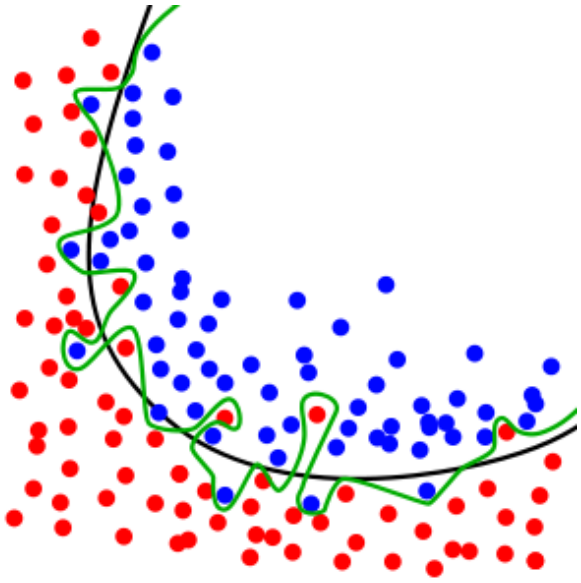


Ilustración 16 Ejemplo de sobreaprendizaje. (Fuente: Wikipedia)

En la Ilustración 16 podemos ver este problema al intentar separar los elementos de dos clases (puntos azules y puntos rojos). La línea negra representa una separación correcta de las clases, aunque se cometan errores en algunos elementos en la frontera. La línea verde representa una separación sobreajustada a los elementos con los que se ha entrenado el modelo.

Una forma de comprobar si nuestro modelo se está sobreajustando al conjunto de entrenamiento consiste en representar el error cometido en la clasificación en cada *epoch,* tanto para el conjunto de entrenamiento como para el de validación. En la Ilustración 17 se puede ver un ejemplo de un modelo en el que el error en entrenamiento se reduce continuamente, pero el error en validación decrece hasta cierto punto en el que vuelve a crecer.

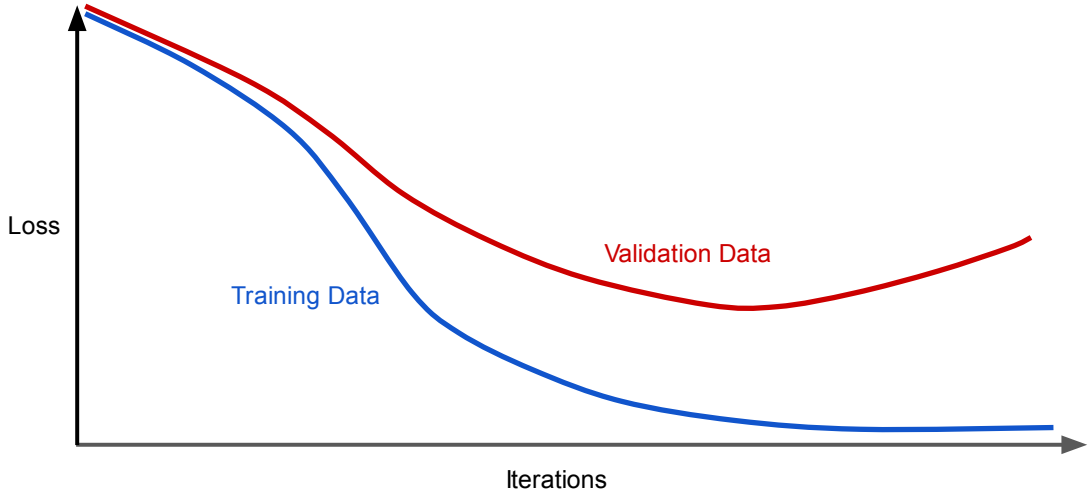


Ilustración 17 Sobreajuste al representar el error con respecto a los epochs. (Fuente: Google Developers)

Existen varias aproximaciones para reducir el sobreajuste de un modelo. En particular, cuando hablamos de CNNs, las soluciones más habituales son:

- **Añadir más datos al conjunto de entrenamiento**. Esta opción es la primera que tenemos que considerar. Añadiendo más datos al conjunto de entrenamiento, el modelo tendrá más ejemplos de los que aprender y no se sobreajustará tan rápidamente. Sin embargo, añadir nuevos datos no siempre es posible, ya sea por el coste (en tiempo, trabajo y/o dinero) que ello supondría o porque simplemente no existen más datos.
- **Utilizar aumento de datos**. El funcionamiento de esta técnica ya se ha explicado anteriormente. Esta es una buena opción cuando no queremos o no podemos añadir más datos, ya que añadiendo más variedad a nuestro conjunto de entrenamiento el valor que obtenemos de cada uno de los ejemplos es mayor.
- **Utilizar técnicas de regularización**. La regularización es un conjunto de métodos que mantienen todos los parámetros del modelo, pero los limitan o reducen de alguna forma. En CNNs el método más utilizado es el *Dropout* [5], ya explicado en este capítulo. Otros métodos populares son la **regularización L1** y la **regularización L2**.
- **Reducir la complejidad de la arquitectura**. Otra forma de reducir el sobreaprendizaje es utilizar un modelo de aprendizaje (una CNN, en este caso) menos compleja, reduciendo la profundidad de la red.

3.2.3. Transferencia de conocimiento

La transferencia de conocimiento (*Transfer Learning* en inglés) es un método utilizado en tareas de aprendizaje automático. Consiste en almacenar el conocimiento obtenido en la resolución de un problema y aplicarlo a otro problema distinto. Por ejemplo, se puede utilizar el conocimiento extraído en un problema de clasificación de coches para clasificar camiones.

Este método es especialmente útil cuando se trabaja con técnicas de aprendizaje profundo, debido a la enorme cantidad de cómputo necesaria para entrenar estas redes neuronales. Utilizando este método podemos aprovechar los parámetros (todos o parte de ellos) que la red ya ha entrenado en la resolución de la primera tarea, para la resolución de una segunda tarea.

Cuando trabajamos con CNNs, lo habitual es reutilizar el conocimiento adquirido en la etapa de extracción de características (capas convolucionales y de submuestreo) de las imágenes y solamente entrenemos con nuestros datos las capas finales (capas completamente conectadas y capa de salida). De esta forma aprovechamos el conocimiento de las imágenes del modelo ya entrenado, mientras que adaptamos la fase en la que se produce la clasificación a nuestro conjunto de clases.

Una de las restricciones que presenta este método tiene que ver con los datos con los que se entrena el modelo. Para poder utilizar la transferencia de conocimiento en la segunda tarea, los datos de entrada a la red tienen que tener las mismas características que los utilizados en la primera tarea. En el caso de las CNNs, si en la primera tarea se ha entrenado la red con imágenes a color RGB de un tamaño determinado, para la segunda tarea las imágenes han de ser también a color RGB y con el mismo tamaño.

Este método puede ser muy útil en aquellas situaciones en las que no dispongamos de muchos datos con los que entrenar nuestro modelo y ya exista un modelo entrenado con datos similares a los nuestros.

3.3. Arquitecturas de CNNs

A lo largo de los años distintos grupos de investigación han creado y probado múltiples arquitecturas de redes neuronales. En particular, dentro del campo de las redes neuronales convolucionales, para la tarea de clasificación de imágenes, muchas de estas arquitecturas se han creado para la competición ILSVRC (*ImageNet Large Scale Visual Recognition Competition).* Esta competición, a la que es común referirse como *ImageNet*, se celebra de forma anual, desde 2010, y en ella se evalúan algoritmos para la clasificación de imágenes y detección de objetos. Los modelos presentados utilizan la base de datos *ImageNet*, que contiene miles de imágenes de objetos.

A continuación, se presentan brevemente algunas de las arquitecturas que han marcado la historia de las CNNs y se detallan las arquitecturas que se van a utilizar en este proyecto.

3.3.1. CNNs clásicas

En el año 1994 Yann LeCun presenta **LeNet5** [4], la que es considerada como la primera red neuronal convolucional. LeNet5 fue la primera red en utilizar la combinación de convolución + submuestreo para la extracción de características de imágenes. Se creó para clasificación de imágenes que contienen números de un solo dígito manuscritos, de 32x32 píxeles.

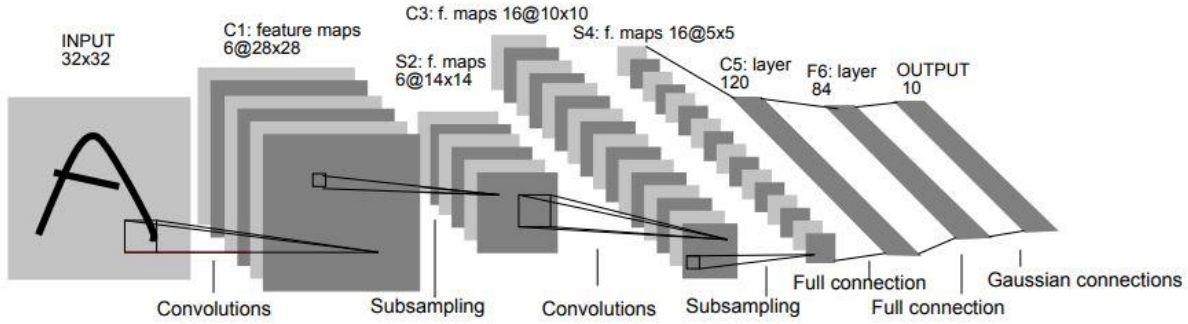


Ilustración 18 Arquitectura de la red LeNet5. (Fuente: [4])

Otro hito en el campo de las CNNs fue la llamada *AlexNet* [6], presentada en el año 2012 por Alex Krizhevsky et al. para la competición ImageNet. La arquitectura es similar a la de LeNet5, aunque considerablemente más grande. Otro de los grandes avances de esta red fue el uso de tarjetas de procesamiento gráfico (*Graphics Processing Unit, GPU*) para acelerar el cómputo de la red.

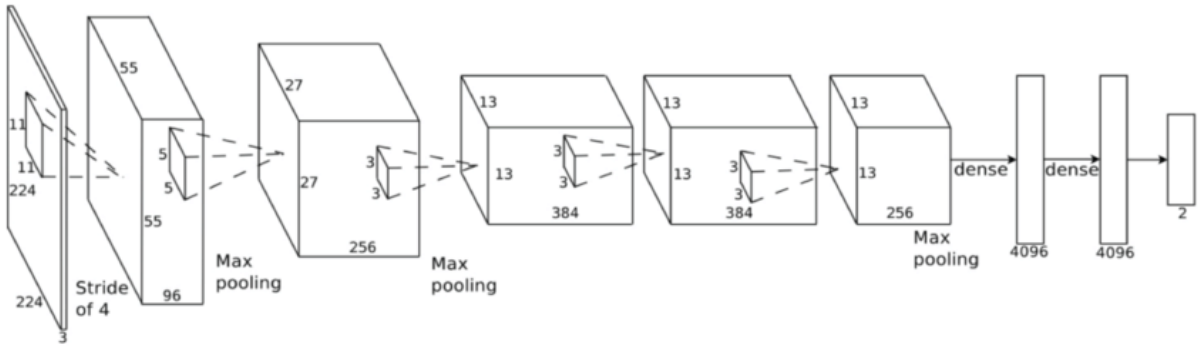


Ilustración 19 Arquitectura de la red AlexNet. (Fuente: [6])

La red **VGG-16** [7], presentada en 2014, también para la competición ImageNet, ofrece una variante más profunda pero más simple que las redes anteriores. En el momento de su publicación se la consideró una red muy profunda.

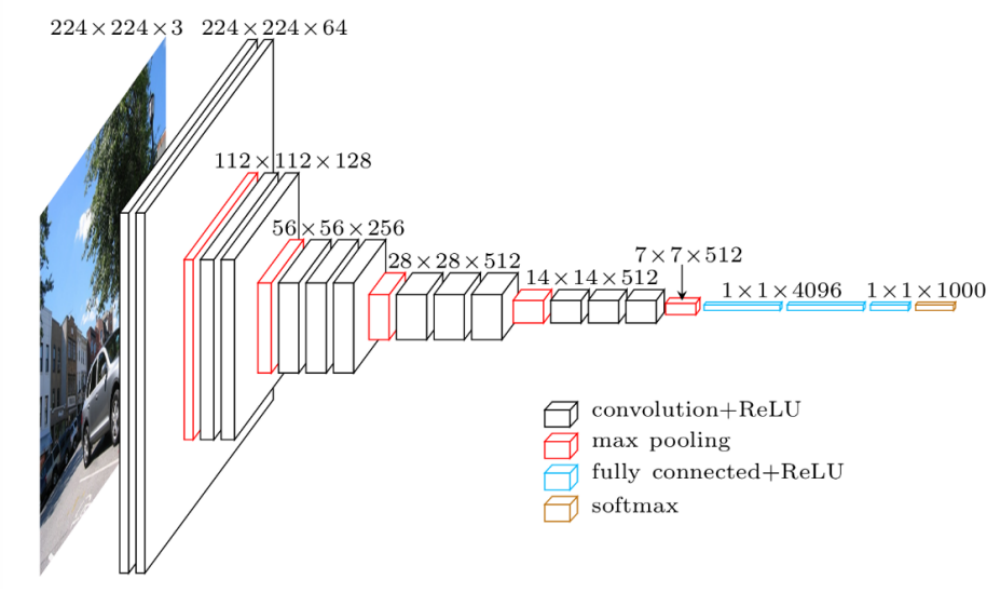


Ilustración 20 Arquitectura de la red VGG-16. (Fuente: [7])

También en 2014 para la competición ImageNet, investigadores de Google presentaron la red *Inception* (o GoogLeNet) [8]. La arquitectura está basada en el **bloque Inception**, que realiza una serie de convoluciones a diferentes escalas y después agrega los resultados. Los filtros utilizados en este bloque son de 1x1, 3x3 y 5x5.

Estos mismos investigadores han publicado varias versiones revisadas de esta red, en las que mejoran la eficiencia en la computación. Una de las innovaciones que presentan es la sustitución de un filtro grande (5x5) por dos filtros sucesivos de tamaño menor (3x3), que obtienen el mismo resultado, utilizando una menor cantidad de parámetros entrenables.

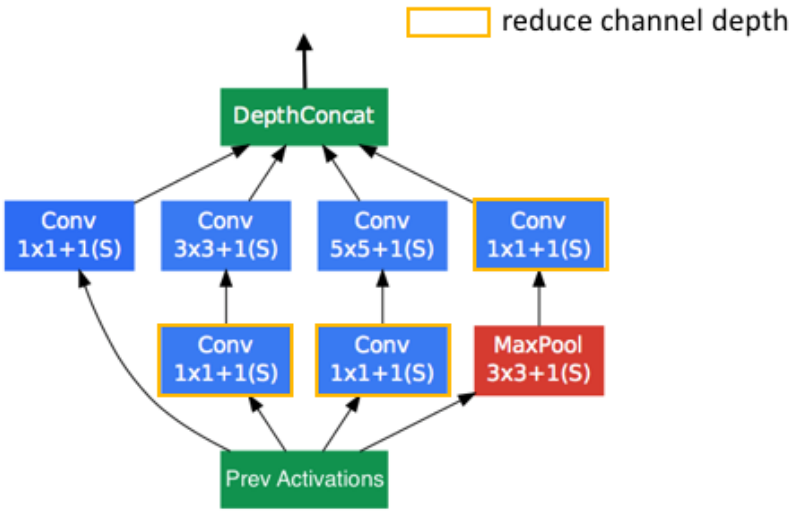


Ilustración 21 Bloque Inception. (Fuente: [8])

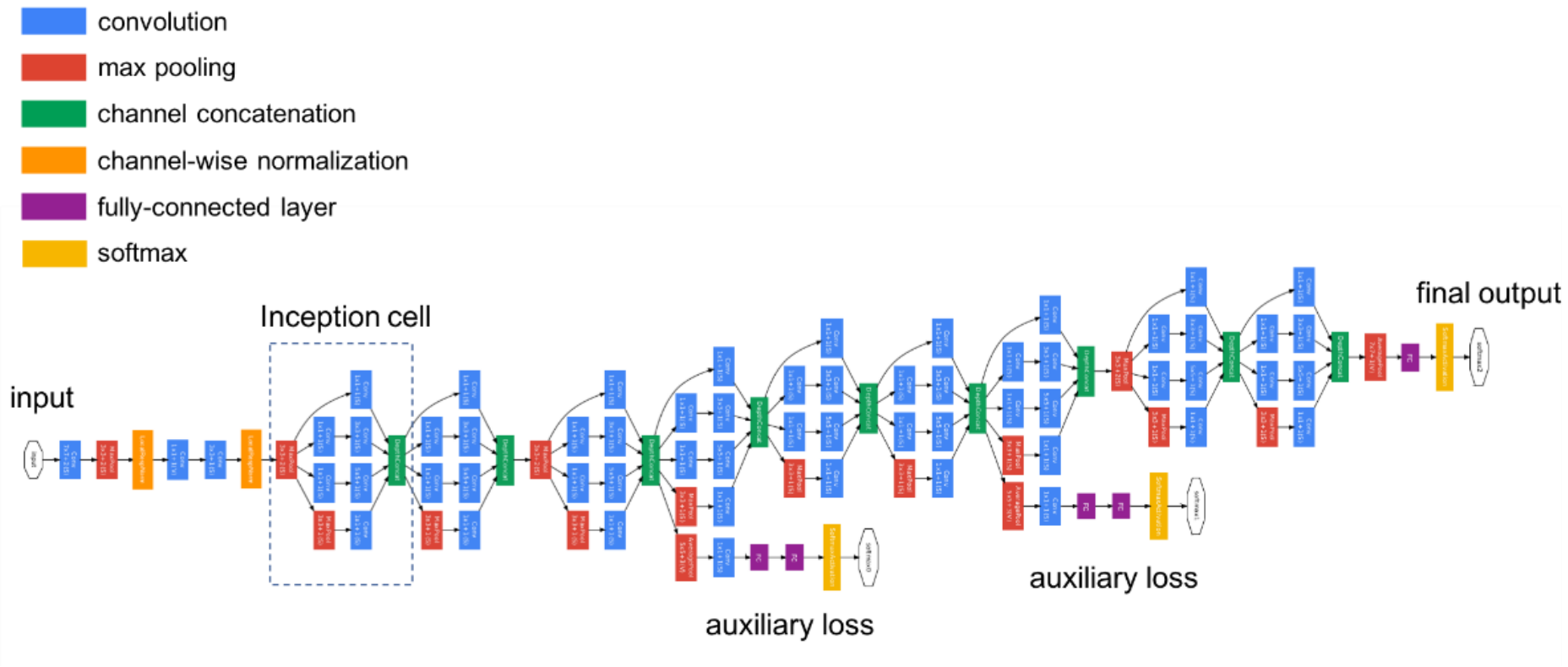


Ilustración 22 Arquitectura de la red Inception (Fuente: [8])

Otro hito en el desarrollo de CNNs es el de las redes profundas residuales (*Deep residual networks*) [10], que permiten el desarrollo de redes mucho más profundas (pasando de decenas de capas a cientos de capas de profundidad).

Estas redes utilizan el llamado **bloque residual** para solventar el problema de la **degradación**, que hace que redes muy profundas tengan un mayor error en entrenamiento que redes menos profundas. En estos bloques residuales, las capas intermedias aprenden una función residual referente a la entrada del bloque.

Existen diferentes versiones de estas redes, como la ResNet 34 o la ResNet 50, que varían principalmente en los hiperparámetros utilizados en las capas de cada bloque residual.

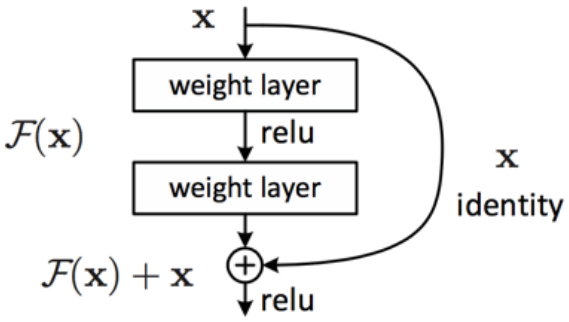


Ilustración 23 Bloque residual. (Fuente: [10])

Ilustración 24 Arquitectura de la red ResNet 34. (Fuente: [10])

Estas no son las únicas arquitecturas de CNNs que existen. Hay disponibles muchas más arquitecturas (como la ResNeXt, una extensión de las redes residuales profundas, o la DenseNet, en la que cada bloque utiliza todos los mapas de características de los bloques anteriores) y surgen más cada año, sin embargo, estas son las más conocidas y populares.

3.4. Tecnologías para Deep Learning

Existen actualmente gran cantidad de recursos para la creación de sistemas *Deep Learning*, principalmente en forma de *frameworks* para diferentes lenguajes de programación. Se presentan a continuación los más utilizados:

- **TensorFlow** [17]**:** biblioteca de bajo nivel de código abierto desarrollada por Google Brain para aprendizaje automático, mediante la construcción y entrenamiento de redes neuronales.

- **Caffe** [19]**:** *framework* para Deep Learning de código abierto, desarrollado originalmente por la Universidad de California, Berkeley. Soporta distintas arquitecturas para la clasificación de imágenes y la segmentación de imágenes.

- **Microsoft Cognitive Toolkit** [20]**:** *framework* para Deep Learning desarrollado por Microsoft Research, permite describir redes neuronales como una serie de pasos mediante un grafo dirigido.

- **Pytorch** [21]**:** biblioteca de código abierto para aprendizaje automático, desarrollada por el grupo de investigación en inteligencia artificial de Facebook.

- **MXNET** [22]**:** *framework* de código abierto para aprendizaje automático, desarrollado por la fundación Apache, para el entrenamiento e implementación de redes neuronales.

- **Chainer** [23]**:** *framework* para Deep Learning de código abierto desarrollado por la compañía japonesa *Preferred Networks*.

- **Keras** [18]**:** biblioteca de código abierto para Deep Learning, diseñada para permitir una experimentación rápida con redes de aprendizaje profundo, capaz de funcionar sobre TensorFlow, Microsoft Cognitive Toolkit o Theano.

4. Diseño, recopilación e implementación de la Base de Datos

En este capítulo se exponen las consideraciones que se han tomado en cuanto al diseño de la base de datos de imágenes, el proceso seguido para obtener estas imágenes, el preprocesamiento que se les ha realizado y la implementación final de la base de datos.

4.1. Consideraciones iniciales

El primer paso en el desarrollo práctico de este proyecto es el de realizar la recopilación, preprocesamiento y etiquetado de las imágenes que conformarán la base de datos.

Esta base de datos ha de cumplir una serie de requisitos para que sea adecuada a los métodos de aprendizaje automático en general, y a los métodos de aprendizaje basados en redes neuronales en particular:

- Ha de representar el mayor porcentaje posible de estilos del tipo de imágenes que estamos tratando (cómics, en nuestro caso).
- Puesto que estamos ante un problema de clasificación, necesitamos varias categorías distintas entre las que poder distinguir.
- Es necesario que la base de datos este conformada por gran cantidad de imágenes, puesto que las redes neuronales necesitan, en principio, una gran cantidad de datos para su entrenamiento.
- Todas las imágenes han de tener el mismo tamaño en cuanto a anchura y altura en píxeles para poder ser procesadas por la red neuronal.

4.2. Revisión de estilos de cómics

Antes de proceder a la recopilación de las imágenes, se ha de realizar una revisión de los distintos estilos de cómics existentes en el mercado, con el fin de obtener una base de datos que represente la mayor cantidad posible de estos estilos.

Tras una exploración de distintas tiendas online especializadas en cómics se ha determinado que existen dos grandes estilos:

- Cómics de estilo occidental: estos cómics o historietas, tanto en blanco y negro como en color, agrupan a todos aquellos creados por autores europeos o americanos.

- Manga: cómics en blanco y negro de procedencia asiática (principalmente japonesa).

Dentro de estos dos grandes estilos existen múltiples subcategorías: manga shojo, manga shonen, cómics de superhéroes, cómics de acción, tebeos, etc.

Para esta base de datos se han obtenido imágenes de aquellas categorías más populares y con mayor variedad de cómics dentro de cada uno de estos dos grandes estilos. Las cinco categorías elegidas son:

1. Manga Shonen: mangas destinados a jóvenes varones de entre 12 y 18 años, caracterizados por grandes cantidades de acción, a menudo con situaciones humorísticas, con protagonistas masculinos.

2. Manga Shojo: mangas dirigidos a un público adolescente femenino. Abarca gran cantidad de temas, desde el drama histórico hasta la ciencia ficción, haciendo énfasis en las relaciones humanas y sentimentales.

3. Cómics de superhéroes: historietas de acción en un marco de ciencia ficción, protagonizadas habitualmente por personajes con poderes sobrehumanos, aunque no necesariamente.

4. Cómics de terror: historietas que pretenden provocar en el lector sensaciones de miedo, pavor, repugnancia u horror. Suelen desarrollar la aparición súbita de alguna fuerza, evento o personaje maligno en un ambiente de normalidad.

5. Historietas tradicionales europeas o tebeos: historietas destinadas a un público infantil que fusiona argumentos de aventuras y cómicos.

4.3. Proceso desarrollado

Una vez elegidas estas categorías se ha procedido a la creación de la base de datos según el siguiente proceso:

1. Recopilación de las imágenes.
2. Preprocesamiento de las imágenes obtenidas.
3. Etiquetado de las imágenes.

Hay que mencionar que el preprocesamiento de las imágenes se ha realizado dos veces para obtener dos versiones finales de la base de datos. Tal y como se ha mencionado en el apartado Transferencia de conocimiento3.2.3 algunas de las

arquitecturas de redes convolucionales utilizadas necesitan que las imágenes sean de un tamaño dado para poder aplicar la técnica de *Transfer Learning*. En concreto, se ha creado una base de datos con imágenes de 135 píxeles de ancho y 225 de alto y otra versión con imágenes de 299 por 299 píxeles.

4.4. Recopilación de imágenes

La recolección de las imágenes se ha realizado mediante un script en Python que permite descargar imágenes haciendo uso de la API de Bing Search Images, englobado en la plataforma Microsoft Azure. Esta API permite la descarga automatizada de gran cantidad de imágenes relevantes de forma rápida, mediante una consulta.

Para hacer uso de esta API hay que crear una cuenta de Microsoft Azure (o utilizar una ya existente) y solicitar la versión de prueba de la API de Bing Search. Esta versión de prueba da acceso a 3,000 descargas (transacciones) al mes, con un límite de 3 transacciones por segundo. Una vez superadas estas 3,000 transacciones, se puede seguir realizando descargas, a un precio de 7$ (5.9€) cada 1,000 transacciones.

Una vez solicitada esta versión de prueba, hemos de obtener nuestras claves de la API y utilizarlas en el *script* de Python para la descarga de las imágenes.

4.4.1. *Script* Python

El fichero `searchBingApi.py` contiene el *script* de Python para realizar la descarga de las imágenes.

En este fichero hay que dar valores a dos cadenas de caracteres:

- *query*: aquí se introduce la consulta que queremos realizar sobre Bing Images.
- *output*: el directorio donde se almacenarán las imágenes.

También hemos de especificar los siguientes parámetros:

- *API_KEY*: nuestra clave para el uso de la API de Bing Search.
- *MAX_RESULTS*: la cantidad máxima de imágenes que queremos descargar.
- *GROUP_SIZE*: el número de imágenes en cada petición (máximo 50).

Una vez introducidos estos parámetros, el programa solicita la cantidad máxima de imágenes a Bing, en varias peticiones de menor tamaño. Se comprueba una a una las imágenes recibidas: si son correctas se escriben en disco, si hay algún problema (se ha capturado alguna excepción) se salta esa imagen y se pasa a la siguiente.

4.4.2. Proceso de recopilación

Utilizando el *script* de Python descrito en el apartado anterior, se ha realizado la descarga de una serie de imágenes de cómics pertenecientes a las categorías descritas en 3.2. Para cada uno de los cómics se han descargado 250 imágenes, que posteriormente se han revisado a mano para eliminar todas aquellas no relevantes (fotografías de películas o juguetes relacionadas con los personajes de los cómics, posters promocionales, etc.).

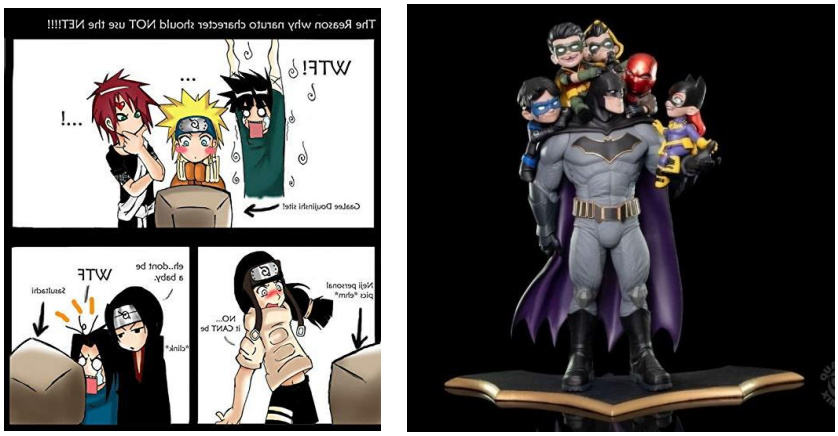


Ilustración 25 Ejemplos de imágenes descartadas.

Para cada uno de estos cómics se ha anotado el nombre del cómic, la categoría a la que pertenece, la editorial que lo publica y el nombre del dibujante.

Los resultados obtenidos se detallan en la siguiente tabla:

Categoría	Comic	Editorial	Dibujante
Manga Shonen	Naruto	Shueisha	Masashi Kishimoto
Manga Shonen	Dragon Ball	Shueisha	Akira Toriyama
Manga Shonen	One Piece	Shueisha	Eiichiro Oda
Manga Shonen	Captain Tsubasa	Shueisha	Yōichi Takahashi
Manga Shonen	Detective Conan	Shueisha	Gosho Aoyama
Superhero	Spider-Man	Marvel	Varios

Superhero	Hulk	Marvel	Varios
Superhero	Batman	DC Comics	Varios
Superhero	Superman	DC Comics	Varios
Superhero	Green Lantern	DC Comics	Varios
Superhero	X-Men	Marvel	Varios
Superhero	Iron Man	Marvel	Varios
Superhero	Wonder Woman	DC Comics	Varios
Tebeo	Astérix	Dargaud	Albert Uderzo
Tebeo	Mortadelo	Bruguera, Ediciones B	Francisco Ibáñez
Tebeo	Spirou	Dupuis	Varios
Tebeo	Tintin	Casterman	Hergé
Tebeo	Zipi y Zape	Bruguera	José Escobar Saliente
Tebeo	Capitán Trueno	Bruguera	Miguel Ambrosio Zaragoza
Manga Shojo	Sailor Moon	Kodansha	Naoko Takeuchi
Manga Shojo	Fruits basket	Hakusensha	Natsuki Takaya
Manga Shojo	Card captor Sakura	Kodansha	Clamp (grupo)
Manga Shojo	Vampire knight	Hakusensha	Matsuri Hino
Manga Shojo	Kitchen princess	Kodansha	Natsumi Andō
Horror	The Walking Dead	Image Comics	Tony Moore
Horror	Hellblazer	DC Comics, Vertigo	Varios
Horror	Tomb of Dracula	Marvel	Varios
Horror	Hellboy	Dark Horse Comics	Mike Mignola

Tabla 2 Información sobre los cómics utilizados para la base de datos. (Fuente: elaboración propia)

El número de cómics para cada categoría se muestra en el siguiente gráfico:

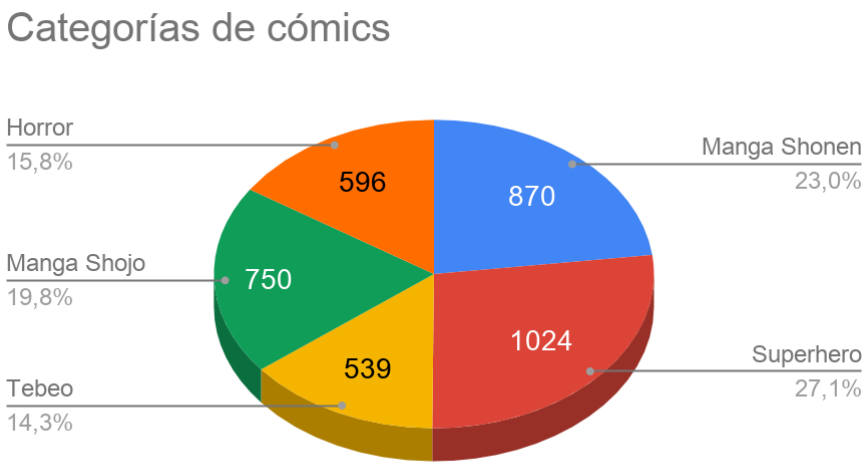


Ilustración 26 Proporción de imágenes por categoría del dataset (Fuente: Elaboración propia).

En total, se han obtenido 3774 imágenes para formar la base de datos.

4.5. Preprocesamiento de las imágenes (versión de 135 x 225 píxeles)

Una vez obtenidas las imágenes, el siguiente paso es redimensionarlas todas a unos mismos valores de ancho y alto, para que puedan ser leídas por una red neuronal. Este redimensionamiento se ha llevado a cabo en dos pasos:

1. Recortado de las imágenes a una relación de aspecto (ancho / alto) común.
2. *Reescalado* o redimensionado de las imágenes a unas mismas dimensiones.

4.5.1. Recortado (*cropping*)

En primer lugar, se ha analizado la relación de aspecto de todas las imágenes en busca de la más frecuente. Este análisis se ha realizado con el *script* de Python `analisisAspectRatio.py`, que obtiene la relación de aspecto (*aspect ratio* en inglés) de todo el dataset y crea un histograma.

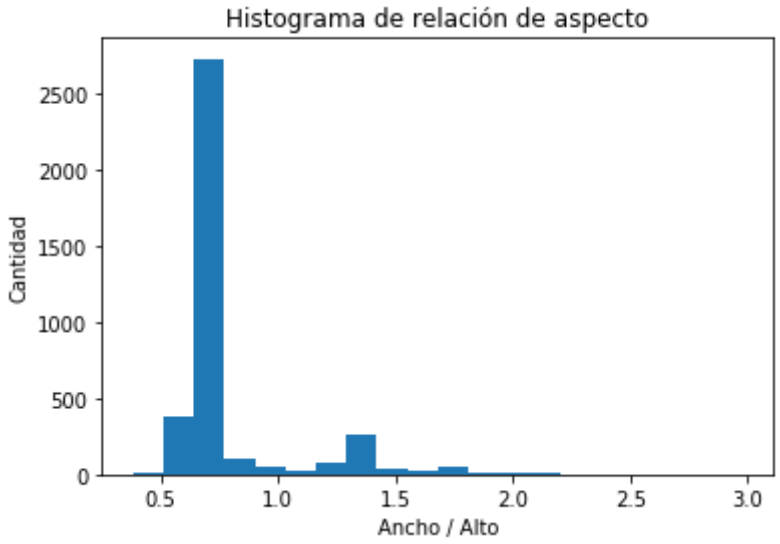


Ilustración 27 Histograma de relaciones de aspecto del dataset descargado. (Fuente: Elaboración propia).

Como se puede comprobar, la *aspect ratio* más común es 0.6, es decir, 3:5, o 3 partes de ancho por cada 5 de alto.

Puesto que esta *ratio* es la más frecuente, con bastante diferencia, todas las imágenes se recortarán para ajustarse a esta *aspect ratio*. Aunque recortar partes de la imagen supone una pérdida de información, es necesario para que las imágenes no se deformen cuando las redimensionemos posteriormente.

Este proceso de recorte (en inglés, *cropping*) consiste en obtener una región rectangular de la imagen, midiendo desde el centro de la misma. El *script* `cropping.py` realiza esta operación leyendo todas las imágenes del *dataset* desde la carpeta “dataset” y recortando la imagen según uno de los siguientes casos:

- Caso A: imágenes cuya relación de aspecto sea inferior a la deseada. En este caso, la anchura se mantiene igual y hemos de obtener la altura correspondiente que mantenga el *aspect ratio* deseado.

$$x = Width$$

$$y = \frac{5 * x}{3}$$

- Caso B: imágenes cuyo aspect ratio sea superior a 1:1, es decir, imágenes más anchas que altas. En este caso, para perder la menor información posible, se gira la imagen 90º, obteniéndose una imagen que puede ser del caso A o del C.

- Caso C: imágenes cuya relación de aspecto esté entre la deseada y 1:1. En este caso, la altura se mantiene igual y hemos de calcular la anchura correspondiente que mantenga el *aspect ratio* deseado.

$$y = Height$$

$$x = \frac{5 * y}{3}$$

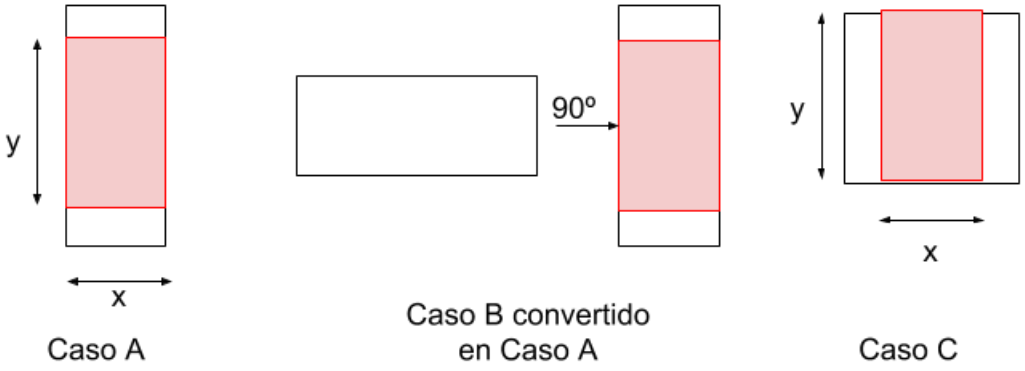


Ilustración 28 Distintos casos de "cropping" (Fuente: Elaboración propia).

Posteriormente se almacena la nueva imagen en la carpeta `datasetCropped`.

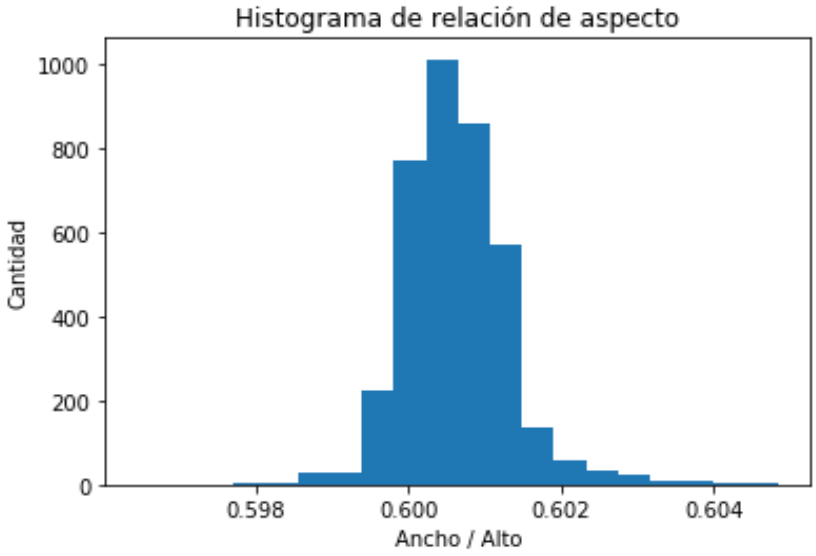


Ilustración 29 Histograma de relaciones de aspecto del dataset recortado. (Fuente: Elaboración propia).

Ahora todas las imágenes tienen la misma relación de aspecto, con minúsculas variaciones (± 0.002), que no influyen en el posterior redimensionado.

4.5.2. Reescalado

Una vez que todas las imágenes tienen la misma relación de aspecto, ya se pueden redimensionar a un mismo tamaño con una menor pérdida de información, siempre que este nuevo tamaño mantenga la relación de aspecto anterior.

Para elegir el tamaño al que vamos a redimensionar las imágenes, vamos a analizar primero cuáles son los valores de ancho y alto más comunes en el dataset tras realizar los recortes.

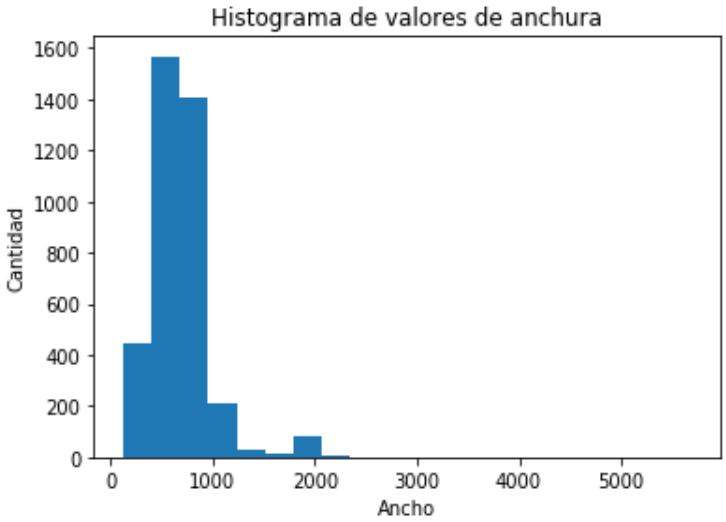


Ilustración 30 Histograma de valores de anchura del dataset recortado. (Fuente: Elaboración propia).

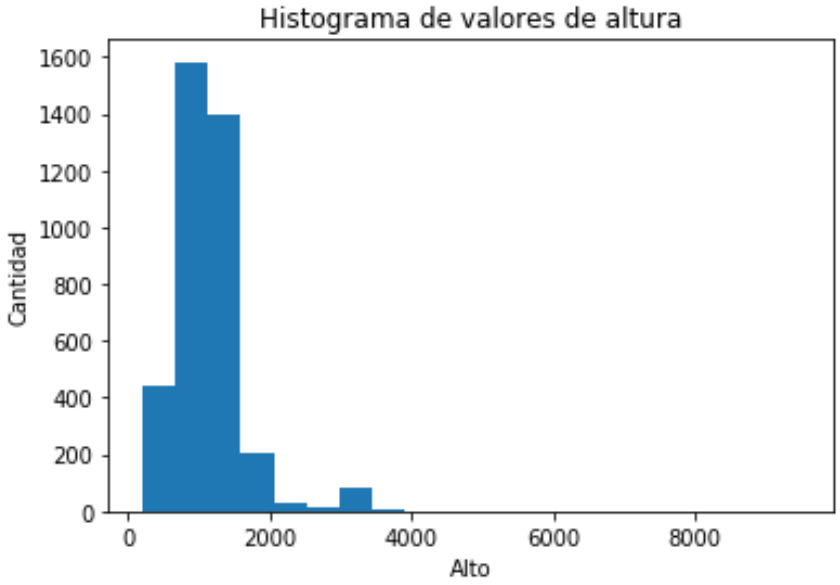


Ilustración 31 Histograma de valores de altura del dataset recortado. (Fuente: Elaboración propia).

Los resultados obtenidos son que, para la anchura, los intervalos con mayor frecuencia son [395, 674) con 1568 apariciones y [674, 953) con 1407. Para la altura obtenemos [658, 1123) con 1583 apariciones y [1123, 1588) con 1396.

Si buscamos un nuevo tamaño que mantenga la relación 3:5 y que sea proporcional a estos valores más frecuentes de ancho y alto encontramos las dimensiones 135 x 225 píxeles. Estas dimensiones, mantienen la relación 3:5 y son, aproximadamente, una quinta parte de los valores 674 x 1123, que son los puntos centrales de los intervalos anteriores de anchura y altura más frecuentes. Con estas dimensiones decididas, podemos llevar a cabo el reescalado del dataset.

Este reescalado se ha realizado con el *script* `resizing.py`, que lee todas las imágenes de la carpeta `datasetCropped`, creando una nueva imagen de tamaño 135 x 225, y almacenándola en la carpeta `datasetResized`. Además, al guardar las nuevas imágenes, se les da un nombre único de cuatro caracteres, del tipo '0000', '0001', etc., que facilitará el posterior etiquetado de las imágenes.

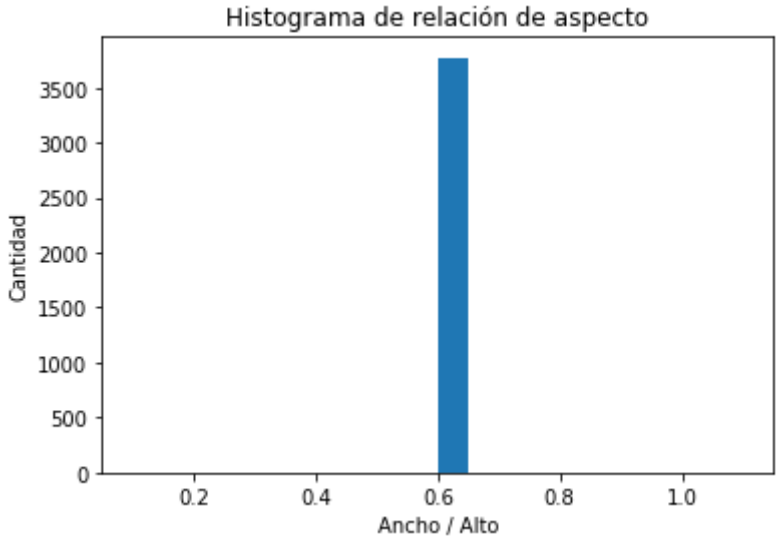


Ilustración 32 Histograma de relaciones de aspecto del dataset redimensionado. (Fuente: Elaboración propia).

4.6. Preprocesamiento de las imágenes (versión de 299 x 299 píxeles)

Para la segunda versión de la base de datos, el preprocesamiento realizado es muy similar, pero no hemos tenido que realizar un análisis de la *aspect ratio*, ya que las imágenes han de ser cuadradas (ratio 1:1) y de tamaño 299 x 299. Esta restricción se debe a que estas imágenes van a ser utilizadas para la técnica de *Transfer Learning*, apartado 3.2.3. Las redes que vamos a utilizar ya han sido entrenadas con imágenes de este tamaño, por lo que si queremos aprovechar los pesos ya preparados de las redes, tenemos que proporcionarles imágenes de este tamaño 299 x 299.

4.6.1. Recortado (cropping)

En esta ocasión, para realizar el proceso de recortado, dado que las imágenes finales han de tener *ratio* 1:1, solamente hemos de considerar los dos siguientes casos:

- Caso A: imágenes cuya relación de aspecto sea inferior a 1:1, es decir, imágenes más altas que anchas. En este caso, la anchura se mantiene igual y hemos de obtener la altura correspondiente que mantenga el *aspect ratio* 1:1.

$$x = Width$$

$$y = x$$

- Caso B: imágenes cuya relación de aspecto sea superior a 1:1, es decir, imágenes más anchas que altas. En este caso, la altura se mantiene igual y hemos de calcular la anchura correspondiente que mantenga la razón 1:1.

$$y = Height$$

$$x = y$$

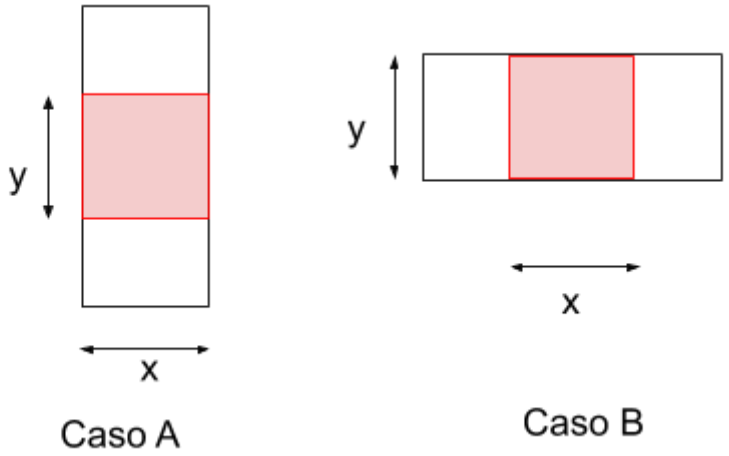


Ilustración 33 Distintos casos de "cropping" (Fuente: Elaboración propia).

Las imágenes recortadas se han almacenado en la carpeta `DatasetCroppedSquare`.

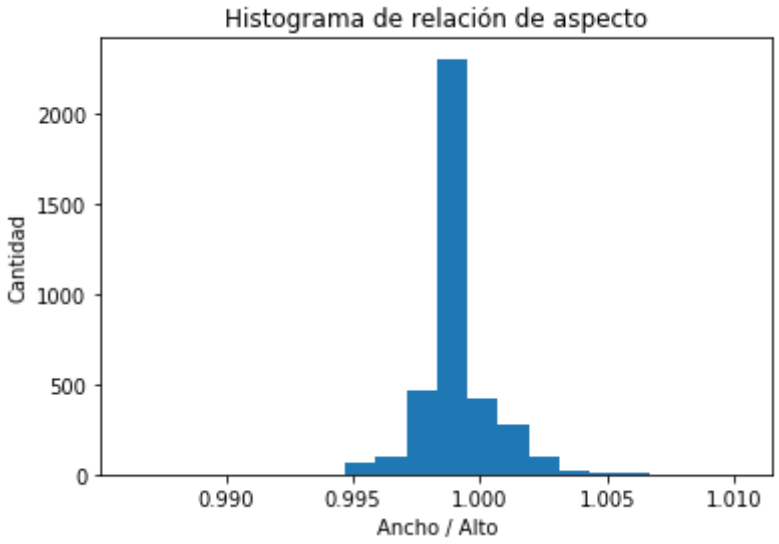


Ilustración 34 Histograma de relaciones de aspecto del dataset recortado. (Fuente: Elaboración propia).

4.6.2. Reescalado

Al igual que para la primera versión de la base de datos, una vez que las imágenes se han recortado a una *aspect ratio* común, se pueden redimensionar sin pérdida de información en cuanto a la relación de aspecto de las imágenes. Para esta versión de la base de datos, de nuevo, no hemos de realizar ningún análisis de cuál es el tamaño más adecuado, ya que sabemos que las arquitecturas que vamos a utilizar necesitan que las imágenes sean de 299 x 299 píxeles.

Utilizando de nuevo el *script* `resizing.py`, se han redimensionado las imágenes de la carpeta `datasetCroppedSquare` y se han almacenado en la carpeta `datasetResizedSquare`. Además, al guardar las nuevas imágenes, se les vuelve a dar un nombre único de cuatro caracteres, del tipo ‘0000’, ‘0001’, etc., que coincide con el nombre dado a la misma imagen preprocesada en la primera versión de la base de datos, simplificando el etiquetado a un único fichero CSV.

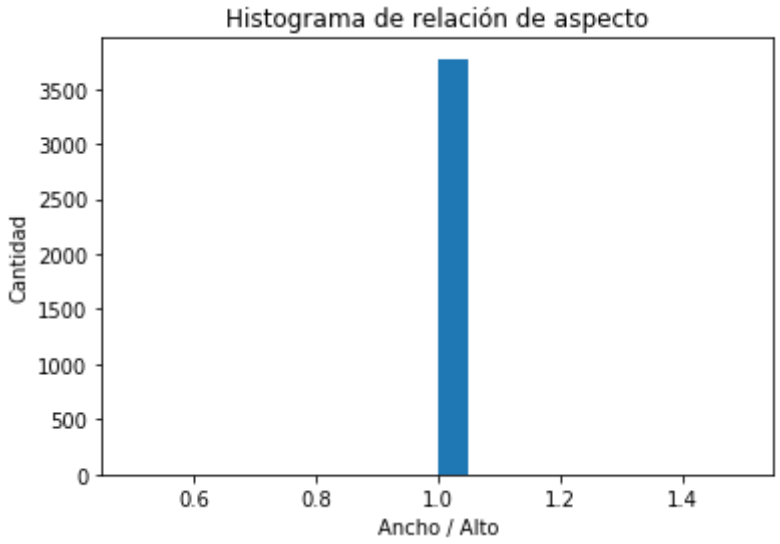


Ilustración 35 Histograma de relaciones de aspecto del dataset redimensionado. (Fuente: Elaboración propia).

4.7. Etiquetado de las imágenes

Una vez que todas las imágenes del conjunto de datos están listas para ser procesadas por una red neuronal, el último paso consiste en etiquetarlas, es decir, indicar a qué categoría pertenece cada imagen.

Este etiquetado se ha llevado a cabo mediante un archivo CSV (*Comma-Separated Values*) con las siguientes columnas: nombre del fichero, categoría a la que pertenece, nombre del cómic, editorial y dibujante.

Este fichero CSV es compartido por ambas versiones de la base de datos, puesto que solamente difieren en el preprocesamiento de las imágenes.

4.8. División de la base de datos

Tal y como se describe en el apartado 2.1.4, una base de datos para Deep Learning se ha de dividir en al menos dos subconjuntos: entrenamiento y test. Para este proyecto se ha utilizado la opción recomendada en la literatura, que consiste en dividir la base de datos en tres subconjuntos, con los siguientes porcentajes de imágenes:

- **Entrenamiento (2264 imágenes, 60% de la base de datos)**. Estas son las imágenes que utilizará la red para aprender las características de las distintas categorías de imágenes.

- **Validación (755 imágenes, 20% de la base de datos)**. Estas imágenes se utilizan para el ajuste de los parámetros de la red, como la configuración de la red, el índice de aprendizaje, etc.

- **Test (755 imágenes, 20% de la base de datos)**. Estas imágenes se utilizan para medir la bondad o porcentaje de acierto de la red cuando se presentan casos (en este caso imágenes) desconocidos.

Una vez dividida la base de datos en estos tres subconjuntos se ha comprobado que todos ellos están balanceados, es decir, mantienen un porcentaje de imágenes de cada categoría similar al de la base de datos completa. En las tres siguientes imágenes vemos estos porcentajes para los tres subconjuntos de la base de datos.

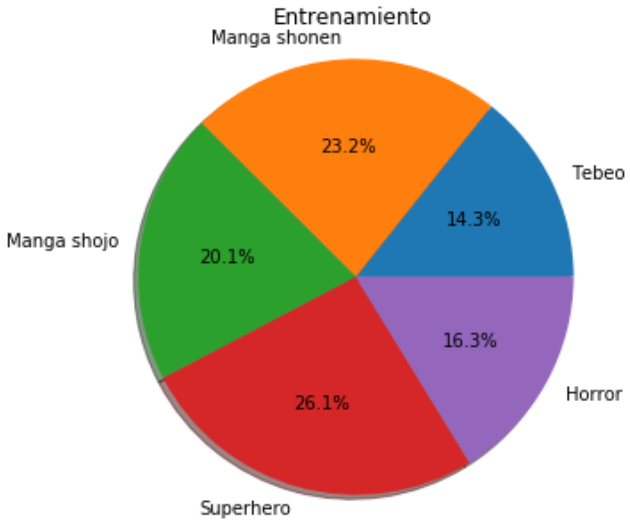


Ilustración 36 Proporción de categorías en el subconjunto de entrenamiento. (Fuente: Elaboración propia).

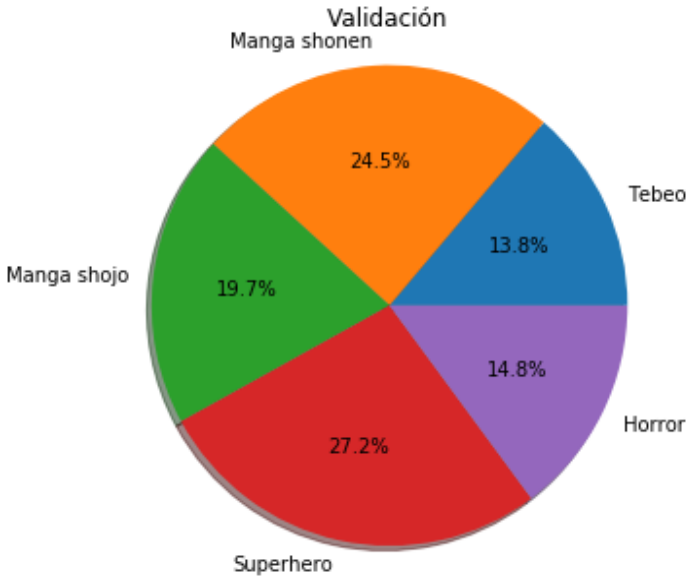


Ilustración 37 Proporción de categorías en el subconjunto de validación. (Fuente: Elaboración propia).

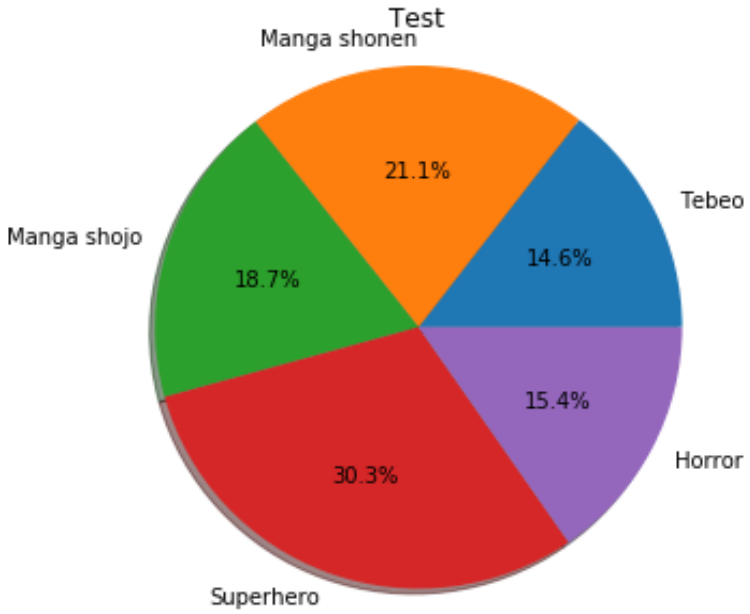


Ilustración 38 Proporción de categorías en el subconjunto de test. (Fuente: Elaboración propia).

	Original	Entrenamiento	Validación	Test
Manga shonen	23.0%	23.2%	24.5%	21.1%
Manga shojo	19.8%	20.1%	19.7%	18.7%
Superhero	27.1%	26.1%	27.2%	30.3%
Tebeo	14.3%	14.3%	13.8%	14.6%
Horror	15.8%	16.3%	14.8%	15.4%

Tabla 3 Proporción de clases en los subconjuntos de la base de datos. (Fuente: elaboración propia)

Tal y como se aprecia en la tabla anterior, los tres subconjuntos presentan un porcentaje de imágenes de cada categoría muy similar (no exactamente igual, pero las diferencias no son significativas) al de la base de datos completa u original.

Al igual que ocurre con el fichero de etiquetado, ambas versiones de la base de datos se han dividido de la misma forma.

4.9. Diagrama Entidad-Relación

El diagrama Entidad-Relación de la base de datos construida es muy simple, puesto que solamente tenemos una Entidad (la imagen), compuesta de 5 atributos: `ID_imagen`, que es la clave primaria, la `Categoría` o estilo del cómic, la `Editorial`, el `Dibujante` y el nombre del `Cómic`.

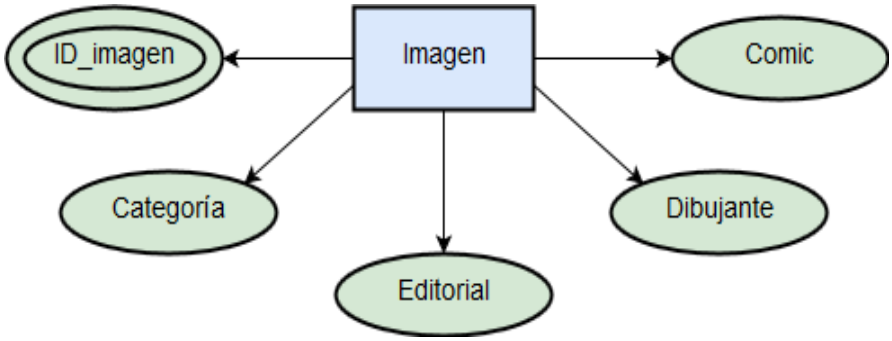


Ilustración 39 Diagrama Entidad-Relación de la base de datos. (Fuente: Elaboración propia).

4.10. Implementación de la Base de Datos

Dado que la base de datos del sistema es muy simple, unas 4000 imágenes distintas, todas pertenecientes a la misma Entidad, no ha sido necesario utilizar un DBMS (*Database Management System* o Sistema Gestor de Bases de Datos), sino que las imágenes se han almacenado directamente en carpetas del sistema de ficheros. Para el índice de la base de datos se ha utilizado el ya mencionado archivo CSV (`etiquetas.csv`), en el que tenemos la información relativa a cada imagen. Dicho fichero sigue el siguiente formato:

1. La primera fila contiene las cabeceras de las columnas, separadas por comas (‘,’). Las columnas siguen el orden: `ID_imagen, Categoría, Cómic, Editorial y Dibujante`.
2. El resto de filas contiene cada una la información relativa a una de las imágenes, siguiendo esta información el mismo orden que las cabeceras:

a. **ID_imagen**: código numérico en formato texto, único para cada imagen, que la identifica. Este código va de '0000' a '3773'.
b. **Categoría**: texto que indica a cuál de las 5 posibles categorías o estilos pertenece la imagen (Tebeo, Superhero, Manga shojo, Manga shonen u Horror).
c. **Cómic**: texto con el nombre del cómic o historieta.
d. **Editorial**: texto con el nombre de la editorial que edita el cómic.
e. **Dibujante**: texto con el nombre del autor del cómic (Varios, si son varios autores).

En el fichero con el código del proyecto, se han incluido las imágenes correspondientes a cada uno de los pasos del preprocesamiento para aquellas personas que quieran acceder a versiones de la base de datos distintas de las versiones finales. Una explicación de los contenidos de cada una de estas carpetas se puede encontrar en el Anexo III.

5. Diseño e implementación del sistema de aprendizaje

En este capítulo se exponen los sistemas de aprendizaje utilizados de acuerdo al problema que pretendemos resolver y los objetivos que queremos alcanzar.

Tal y como se ha explicado anteriormente, el sistema que vamos a desarrollar tiene como objetivo identificar el estilo de dibujo de imágenes de cómics de entre unos estilos predefinidos. Se trata claramente de un problema de clasificación, en el que la propia imagen es la información de entrada al sistema y el estilo de dibujo o tipo de cómic, la salida.

Otro de los objetivos del proyecto es el estudio de redes neuronales convolucionales (CNNs), utilizando técnicas de aprendizaje profundo (Deep Learning), para resolver este problema de clasificación de imágenes.

Una vez definidos estos aspectos del problema, solamente resta por definir la arquitectura o arquitecturas de CNNs que vamos a utilizar y las tecnologías que utilizaremos para construir el modelo.

5.1. Arquitecturas utilizadas

Como se ha explicado en el capítulo 1 la gran mayoría de las arquitecturas de CNNs más populares de los últimos años han sido diseñadas para trabajar con la base de datos de ImageNet. Esta base de datos cuenta actualmente con más de 14 millones de imágenes etiquetadas en más de 20000 categorías. Nuestra base de datos, con algo menos de 3800 imágenes repartidas en 5 categorías, es, en comparación, diminuta.

No entra dentro de los objetivos de este proyecto la creación de una nueva arquitectura para trabajar con esta base de datos, por lo que nos limitaremos a utilizar algunas de las arquitecturas ya diseñadas y probadas.

Se han seleccionado tres arquitecturas con las que, en sucesivas rondas de pruebas, buscaremos el modelo que mejor resultados nos proporcione para nuestra base de datos.

En primer lugar, se ha seleccionado la arquitectura de la red VGG-16, la versión más simple de una de las primeras redes en ser consideradas “muy profundas”.

La siguiente arquitectura seleccionada es la llamada Inception V3, que, aún siendo mucho más profunda que las redes VGG, utiliza muchísimos menos parámetros entrenables que estas y obtiene (en la competición de ImageNet) mejores resultados.

Por último, se ha seleccionado la arquitectura Inception ResNet, que combina las ideas de la red Inception con las redes neuronales residuales.

A continuación, se presentan en mayor detalle estas arquitecturas utilizadas en el proyecto.

3.1.1. VGG-16

La red VGG-16 es una red neuronal convolucional presentada por Karen Simonyan y Andrew Zisserman [7] para la competición *ImageNet* en el año 2014. Estos investigadores presentaron varias redes, siguiendo la misma arquitectura, pero variando el número de capas convolucionales, desde la VGG-11, hasta la VGG-19.

La arquitectura de la red, en todas las versiones, parte de una imagen de tamaño 224 x 224, que pasa por 5 etapas de extracción de características hasta reducirse a 7x7. A continuación, el resultado anterior pasa por tres capas completamente conectadas y, por último, por una capa de salida *softmax.*

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Ilustración 40 Distintas configuraciones de la arquitectura VGG. (Fuente: [7])

De las diferentes configuraciones posibles de esta arquitectura, se ha seleccionado la configuración D (VGG-16), que contiene 16 capas de parámetros entrenables, con un total de 138 millones de parámetros.

3.1.2. Inception (v3)

La arquitectura de CNNs *Inception* [8] fue propuesta por investigadores de Google para la competición *ImageNet*. Uno de los objetivos de esta arquitectura es alcanzar resultados similares o mejores a los de las redes VGG, pero con una menor cantidad de parámetros. La arquitectura de la red está basada en el bloque *Inception (*Ilustración 21*),* de diseño muy complejo, que utiliza múltiples "trucos" para mejorar la eficiencia de la red, tanto en precisión como en velocidad. Las distintas versiones de estos bloques dan lugar a las distintas versiones de la red. En este proyecto utilizaremos la versión 3 [9], que sustituye los filtros de gran tamaño de la red original (7x7 o 5x5) por múltiples filtros consecutivos de menor tamaño. Este proceso se conoce como **factorización.**

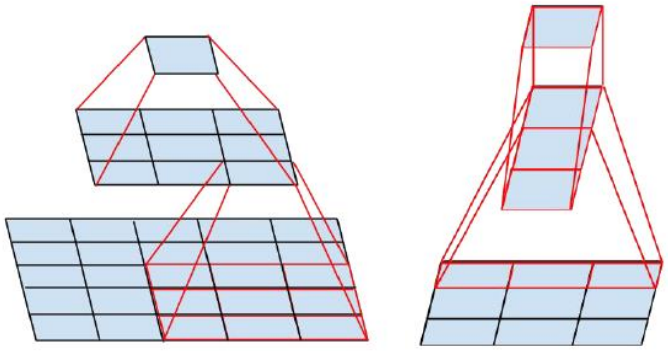


Ilustración 41 Factorización de un filtro 5x5 (Fuente: [9])

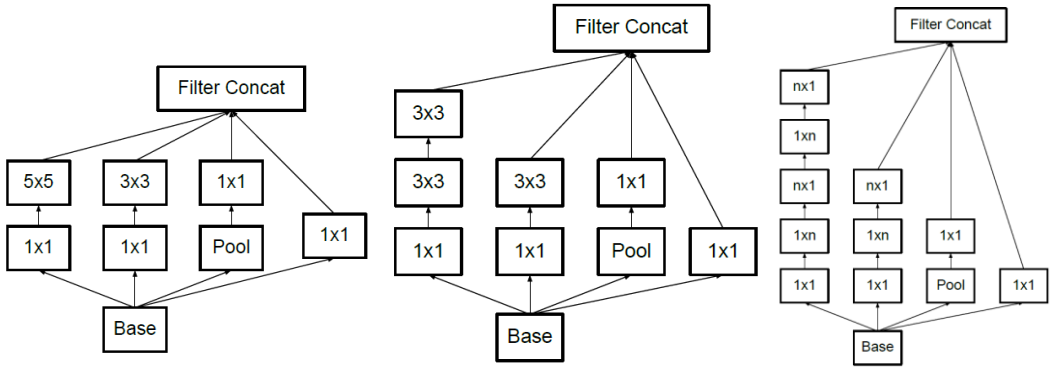


Ilustración 42 Evolución del bloque Inception desde la V1 (izqda.) hasta la V3 (dcha.). (Fuente: [9])

La versión 3 de la red Inception obtiene unos resultados mejores a los obtenidos por la red VGG para el conjunto de datos *ImageNet*, utilizando muchos menos parámetros: 23 millones de parámetros de la red Inception frente a los 138 millones de la red VGG.

3.1.3. InceptionResNet

La última arquitectura utilizada en este proyecto es la arquitectura InceptionResNet [11], con la que los desarrolladores de la arquitectura Inception original combinaron esta con la arquitectura de las redes neuronales residuales. Para ello utilizan un bloque *Inception* híbrido.

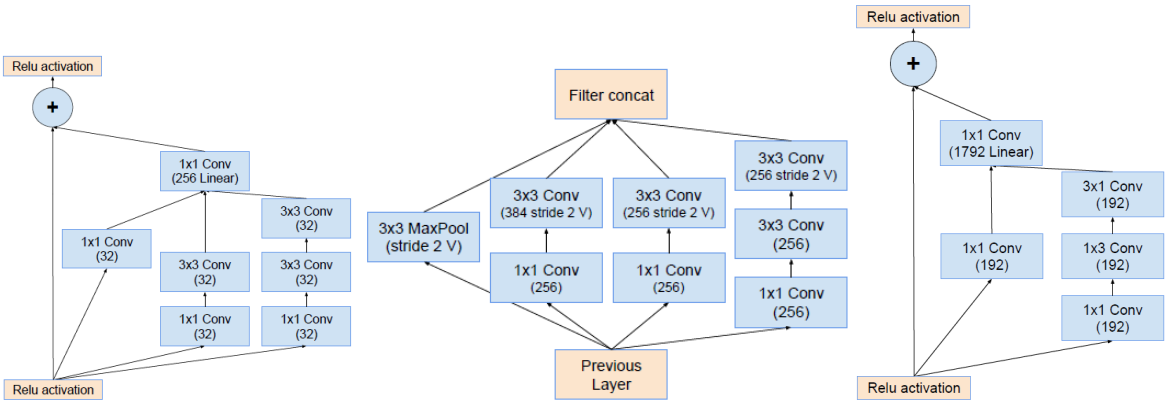


Ilustración 43 Distintos bloques Inception utilizados en la red InceptionResNet. (Fuente: [11])

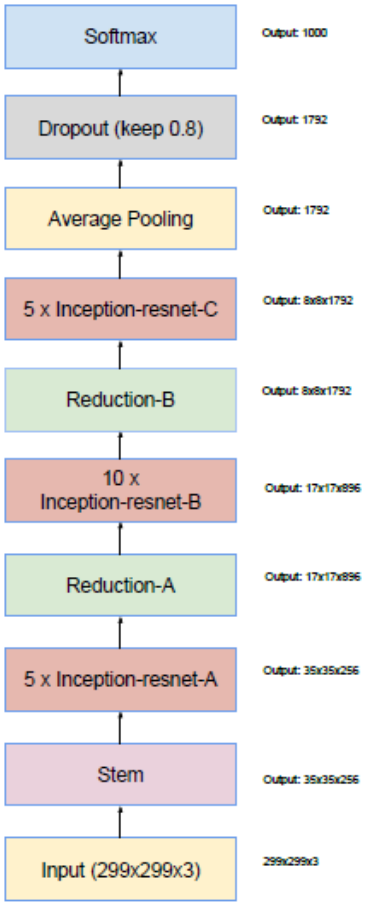


Ilustración 44 Esquema de la red InceptionResNet. (Fuente: [11])

Esta red supera los resultados obtenidos por la red Inception, utilizando, aproximadamente, 56 millones de parámetros.

5.2. Tecnologías utilizadas

En el apartado 3.4 describíamos los *frameworks* más utilizados actualmente para la implementación de redes neuronales. Antes de elegir uno de estos *frameworks*,

veamos una comparación de ellos en algunos aspectos importantes: la plataforma sobre la que trabajan, el lenguaje de programación que soportan como interfaz, si permiten aceleración mediante GPU (CUDA) y si permiten construir redes convolucionales.

Framework	Plataforma	Interfaz	Soporte Cuda	Redes convolucionales
TensorFlow	Windows, macOs, Linux	Python (Keras), C/C++, Java, Go, JavaScript, R, Julia, Swift	Sí	Sí
Caffe	Windows, macOs, Linux	Python, Matlab, C++	Sí	Sí
Microsoft cognitive toolkit	Windows, macOs (vía Docker), Linux	Python (Keras), C++, Command line, BrainScript (.NET on roadmap)	Sí	Sí
Pytorch	Windows, macOs, Linux	Python	Sí	Sí
MXNET	Linux, macOS, Windows, AWS, Android, iOS, JavaScript	C++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl	Sí	Sí
Chainer	Windows, macOs, Linux	Python	Sí	Sí
Keras	Windows, macOs, Linux	Python, R	Sí	Sí

Tabla 4 Comparativa de frameworks de Deep Learning. (Fuente: Elaboración propia. Datos: Wikipedia)

Como se puede comprobar, todas las grandes alternativas que existen en el mercado nos permiten crear redes convolucionales para clasificación de imágenes, con soporte en CUDA para acelerar el aprendizaje haciendo uso de la GPU.

En la siguiente gráfica podemos ver una comparativa de estos (y otros) *frameworks* en base a diversos factores como número de contribuciones en Github, número de búsquedas en Google, etc., publicado en la web Towards Data Science.

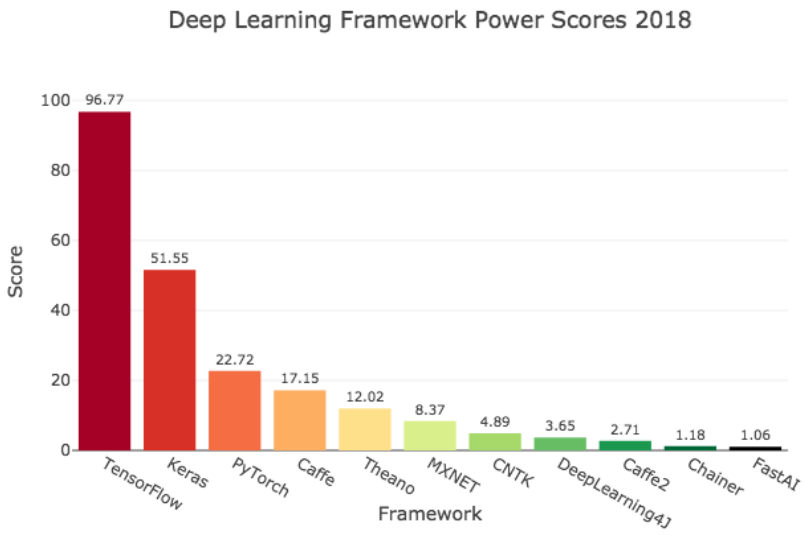


Ilustración 45 Comparativa de frameworks Deep Learning. (Fuente: Towards Data Science [16])

De acuerdo con los resultados mostrados en el artículo [16], TensorFlow es, con mucha diferencia (96.77 puntos sobre 100), el framework más utilizado, seguido por Keras (51.55) y PyTorch (22.72) y Caffe (17.15).

Hemos de tomar una decisión con respecto a qué framework utilizar: ¿TensorFlow o Keras?

Para este proyecto vamos a seleccionar **Keras**, que nos permite crear redes convolucionales que funcionen sobre TensorFlow. La gran ventaja de utilizar Keras en lugar de TensorFlow es su interfaz simple, con un alto nivel de abstracción, que nos permite un prototipado rápido. Puesto que estas redes funcionan sobre TensorFlow y hemos de instalarlo de igual forma en el equipo, siempre podemos saltarnos la capa de abstracción que proporciona Keras y trabajar directamente sobre TensorFlow.

6. Pruebas realizadas y resultados

En este capítulo se exponen las pruebas o experimentos realizados, justificando el proceso seguido para alcanzar el modelo final que mejor se ajusta a nuestra base de datos.

En primer lugar, se han realizado una serie de pruebas preliminares, utilizando las arquitecturas descritas en el punto 5.1, con el objetivo de obtener una visión de cómo se comportan cada una de estas redes ante las imágenes de la base de datos creada. En esta etapa se han realizado varias pruebas con cada una de las arquitecturas, realizando pequeñas modificaciones en los hiperparámetros de la red.

Una vez finalizadas estas pruebas, se han estudiado los resultados obtenidos a fin de seleccionar aquella arquitectura que nos proporcione los mejores resultados para nuestros datos.

A continuación, habiendo seleccionado esta arquitectura, se ha realizado una búsqueda en cuadrícula para ajustar los hiperparámetros que nos optimizan la clasificación de las imágenes.

Finalmente, utilizando la arquitectura y los hiperparámetros anteriormente seleccionados, se ha realizado una última prueba con el fin de obtener el porcentaje de acierto de la red ante datos que nunca había visto, el conjunto de prueba o test.

6.1. Pruebas preliminares

Tal y como se ha mencionado arriba, las primeras pruebas realizadas tienen como objetivo darnos una visión de cómo se comportan las distintas arquitecturas ante los datos con los que vamos a trabajar.

Primeramente, se han entrenado las tres arquitecturas durante 100 *epochs*, utilizando las 2264 imágenes del conjunto de entrenamiento para aprender las características de las imágenes y las 755 imágenes del conjunto de validación para ajustar los parámetros. Otros hiperparámetros utilizados, sin modificar durante estas pruebas preliminares, son:

- Como **optimizador** se ha utilizado el método *Stochastic Gradient Descent.*
- El **tamaño de lote** (*batch size*) usado ha sido de 16.

- Como métrica, dado que simplemente queremos una visión del funcionamiento de la red, solamente hemos utilizado la **precisión predictiva** (*accuracy*).

En las tres imágenes siguientes podemos ver los resultados obtenidos. Las gráficas muestran la precisión en entrenamiento (rojo) y en validación (azul) de la red a lo largo de los 100 *epochs*.

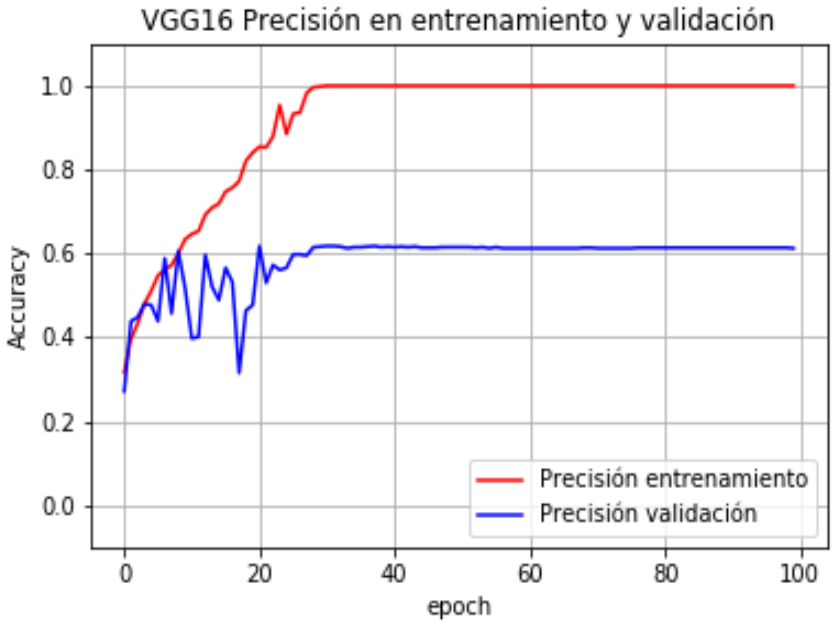


Ilustración 46 VGG 16. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

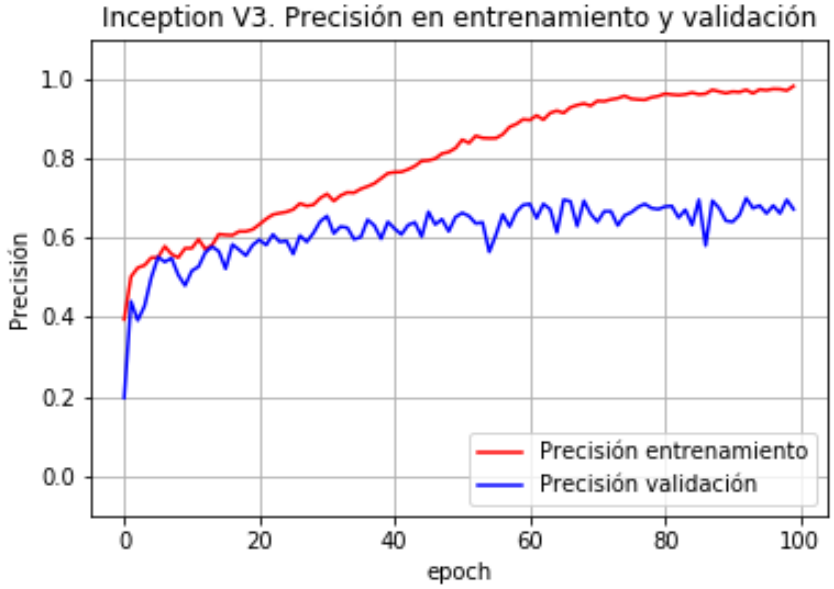


Ilustración 47 Inception V3. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

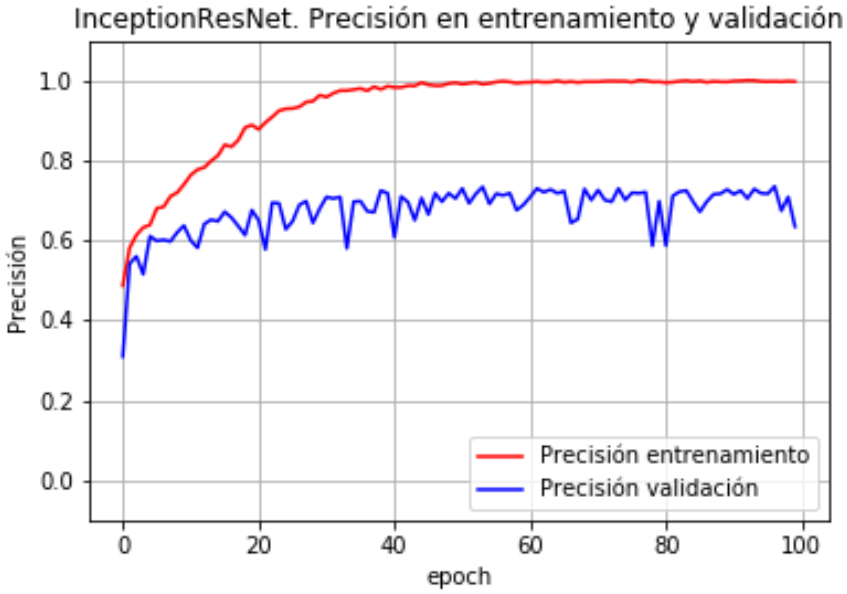


Ilustración 48 InceptionResNet. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

Podemos observar que todas las arquitecturas alcanzan una precisión cercana al 100% en los datos de entrenamiento, pero en los datos de validación esta precisión alcanza un punto en el que deja de mejorar, nunca superando una precisión de en torno al 70%. Estamos claramente ante un problema de sobreajuste (*overfitting*). En el punto 3.2.2, vimos varias opciones posibles para solucionarlo:

- Utilizar más datos.
- Aplicar *Dropout*, o alguna otra técnica de regularización.
- Utilizar aumento de datos.
- Reducir la complejidad de la red.
- También la técnica de la transferencia de conocimiento (*Transfer Learning*) podría ser útil.

A continuación, vamos a aplicar algunas de estas técnicas, para intentar mejorar los resultados obtenidos.

4.1.1. Pruebas preliminares con Dropout

Comenzamos aplicando la técnica de *Dropout*, añadiendo a la última capa completamente conectada de cada una de las redes un valor de **0.5** al hiperparámetro del *dropout* de dicha capa. Esto hará que, en cada *epoch,* la mitad de las neuronas de esta capa, seleccionadas de forma aleatoria, se desactiven.

En las tres siguientes gráficas vemos los resultados obtenidos.

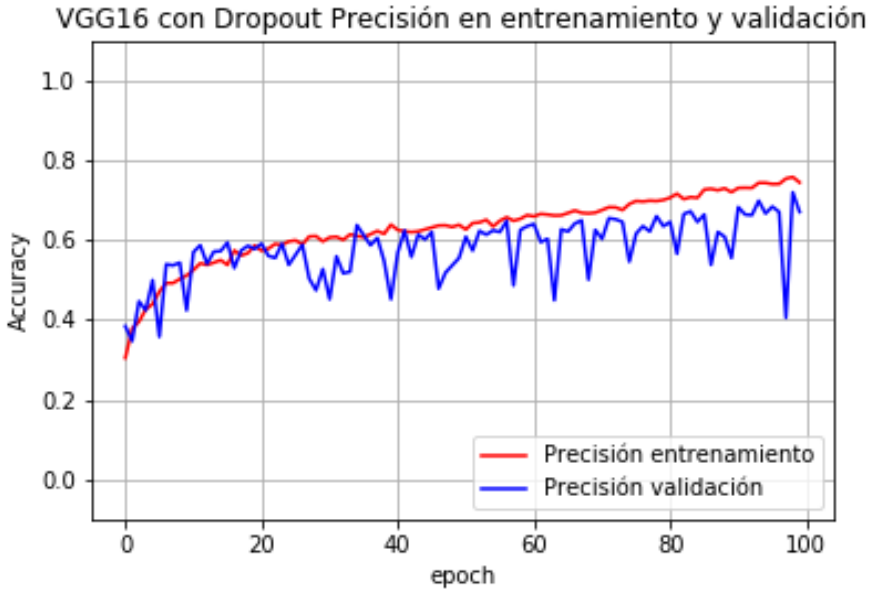


Ilustración 49 VGG 16 con Dropout. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

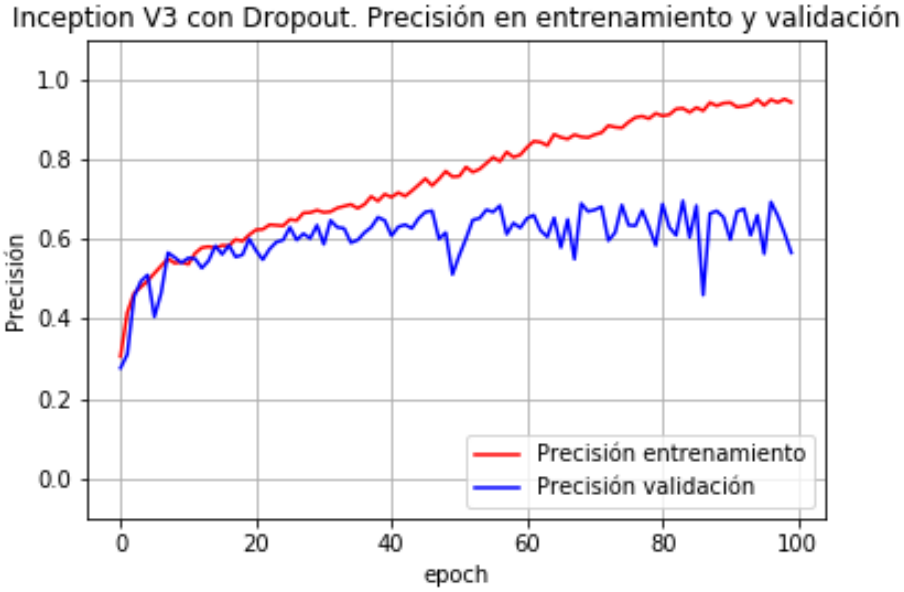


Ilustración 50 Inception V3 con Dropout. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

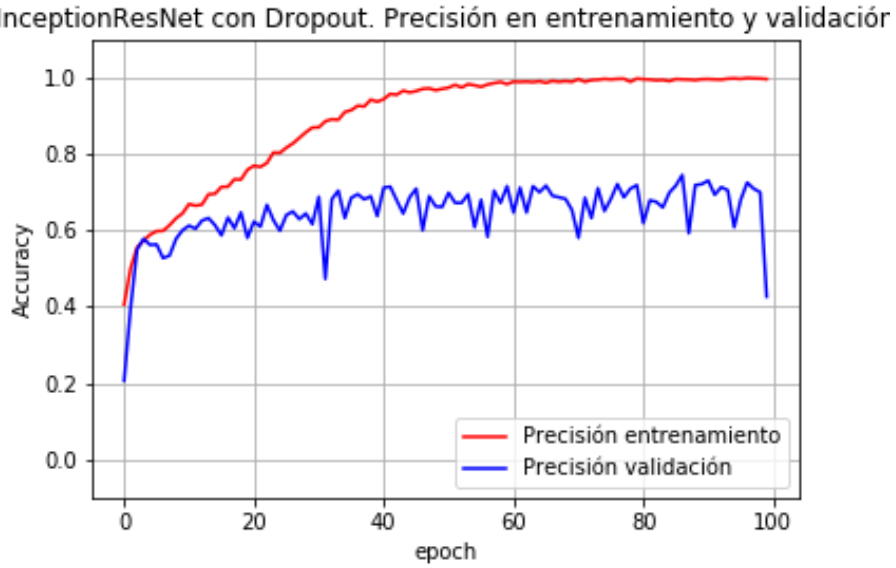


Ilustración 51 InceptionResNet con Dropout. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

Los resultados muestran que, conforme aumenta la complejidad de la red (siendo VGG-16 la red más simple e InceptionResNet la más compleja), menor efecto tiene este *dropout*. En la Ilustración 49 vemos que para la red VGG-16, las dos curvas se mantienen bastante cercanas a lo largo del entrenamiento de la red, mientras que para ambas versiones de Inception las curvas comienzan creciendo una cerca de la otra, pero se separan tras un cierto número de *epochs,* creciendo la precisión en el entrenamiento hasta el 100%, pero estancándose la de validación en torno al 70%.

Estos resultados indican que en la red VGG-16 este *dropout* ha servido para reducir el sobreajuste de la red, mientras que para las arquitecturas Inception, bastante más complejas, este *dropout* no es suficiente.

4.1.2. Pruebas preliminares con *Data Augmentation*

La siguiente técnica de reducción de sobreajuste que vamos a probar es el **aumento de datos** o *data augmentation.* Para ello vamos a utilizar las funciones de preprocesamiento de imágenes que incorpora Keras, realizando a las imágenes de entrenamiento las siguientes modificaciones de forma aleatoria:

- Rotaciones de hasta **40º**.
- Traslaciones para descentrar las imágenes de hasta un **20%** del tamaño de la imagen.
- *Shear* de hasta un 20% del tamaño de la imagen.

- *Zoom* o aumento de hasta el 20% del tamaño de la imagen.
- Volteado horizontal de la imagen.

Además de este aumento de datos, se han probado dos versiones para cada red, una con el mismo *dropout* utilizado en el apartado anterior y otra sin *dropout.*

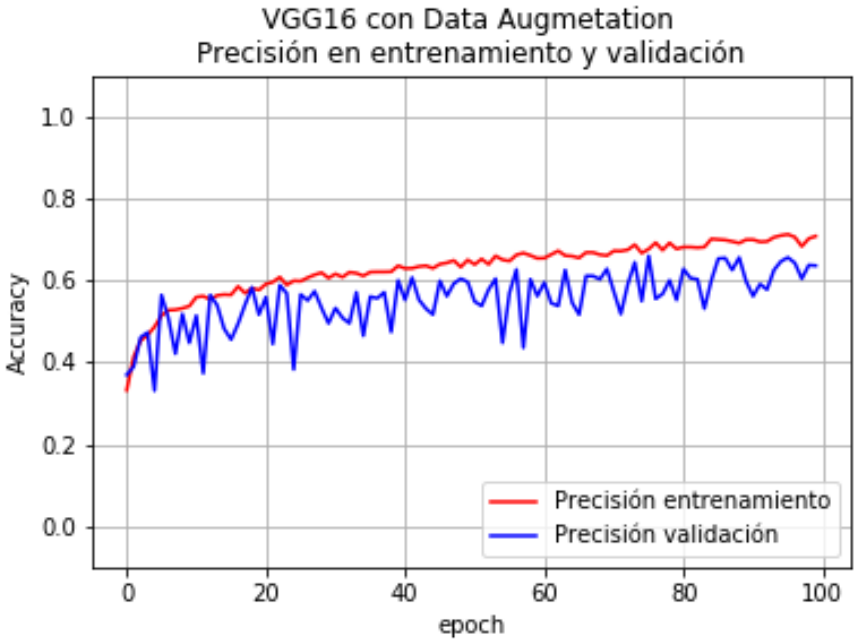


Ilustración 52 VGG-16 con data augmentation. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

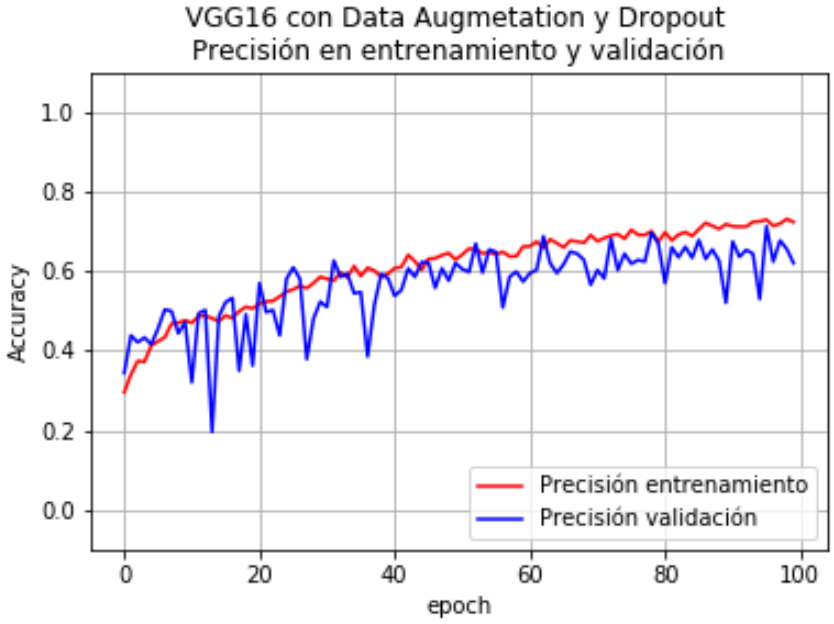


Ilustración 53 VGG-16 con data augmentation y Dropout. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

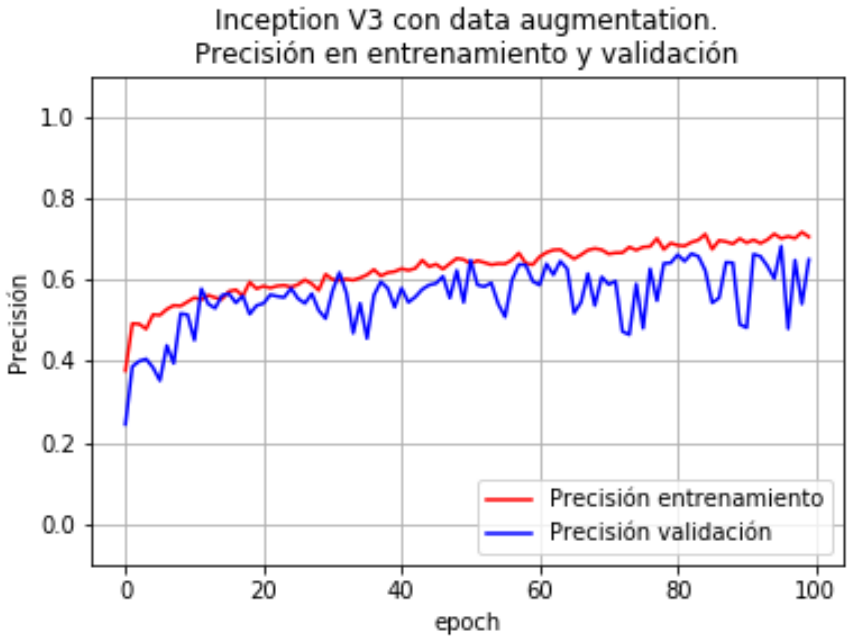


Ilustración 54 Inception V3 con data augmentation. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

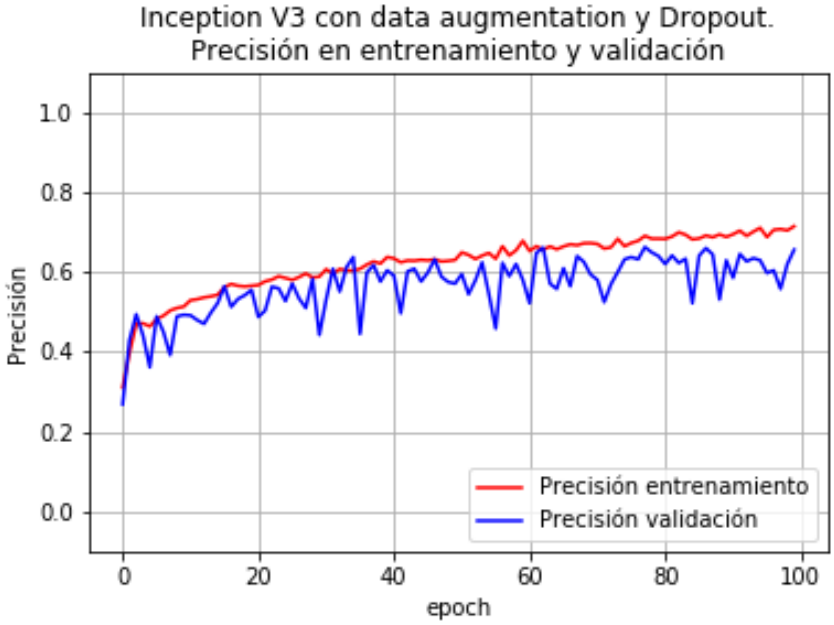


Ilustración 55 Inception V3 con data augmentation y Dropout. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

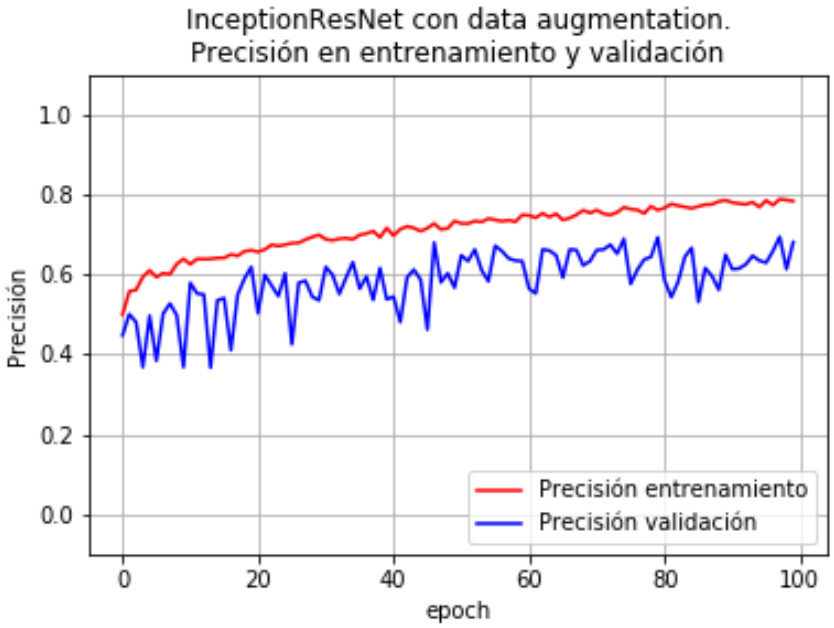


Ilustración 56 Inception V3 con data augmentation. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

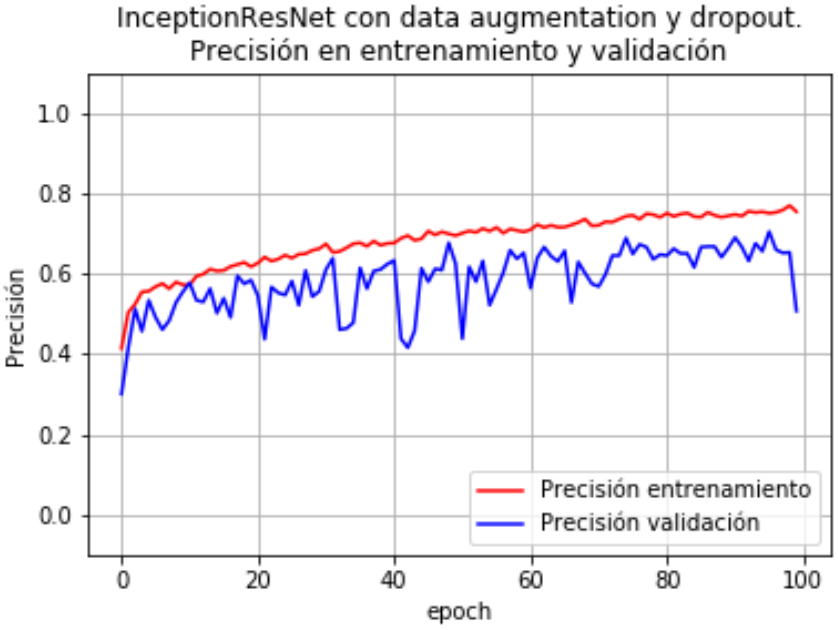


Ilustración 57 Inception V3 con data augmentation y Dropout. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

Los resultados muestran que esta técnica funciona bastante bien con las tres arquitecturas a la hora de reducir la diferencia entre las curvas de precisión en entrenamiento y en validación, pero no consigue mejorar la precisión del modelo.

4.1.3. Pruebas preliminares con transferencia de conocimiento

La siguiente técnica que vamos a probar es la transferencia de conocimiento (*transfer Learning*) que, aunque no es una técnica de reducción del sobreajuste *per se,* puede dar buenos resultados en las circunstancias adecuadas.

Para utilizar esta técnica vamos a entrenar las redes Inception e InceptionResNet[3] con las imágenes de 299 x 299 píxeles, fijando los pesos del *dataset ImageNet* en las capas de extracción de características y entrenando el resto de capas completamente conectadas. Al igual que en las pruebas anteriores, se ha entrenado una versión de cada red con *Dropout* y otra sin él.

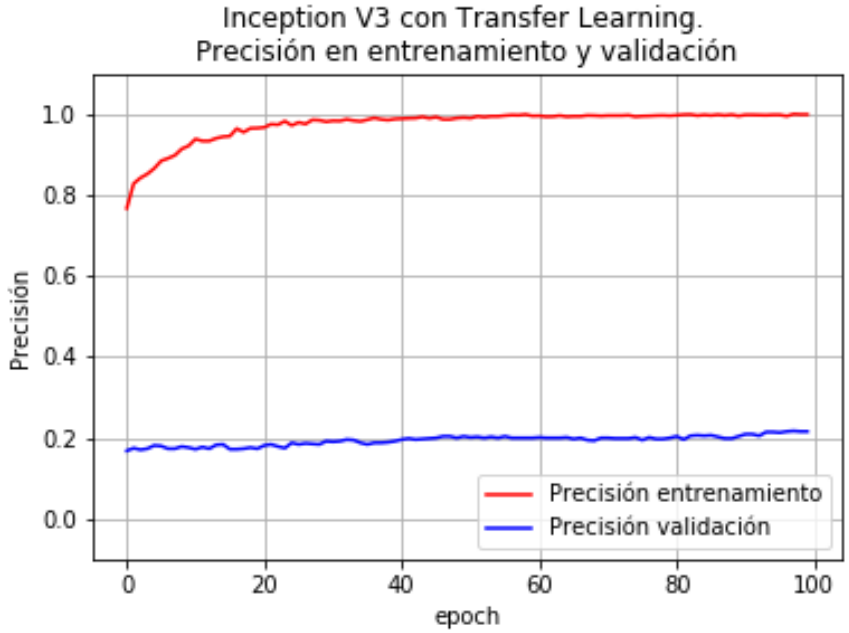


Ilustración 58 Inception V3 con Transfer Learning. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

[3] La red VGG-16 necesita imágenes de 224 x 224 píxeles para poder utilizar la transferencia de conocimiento. No tenemos en nuestra base de datos imágenes de este tamaño y no se ha construido una nueva versión de esta por falta de tiempo.

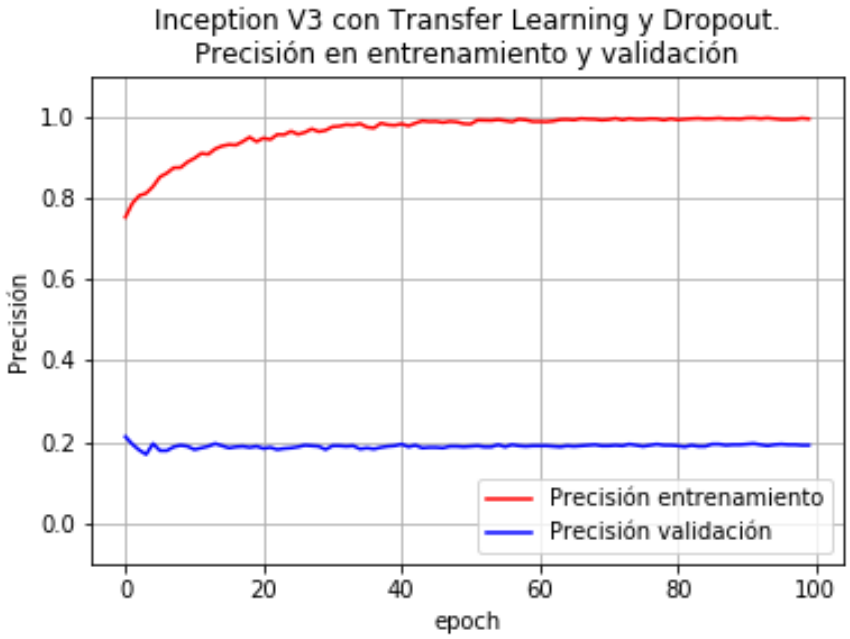


Ilustración 59 Inception V3 con Transfer Learning y Dropout. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

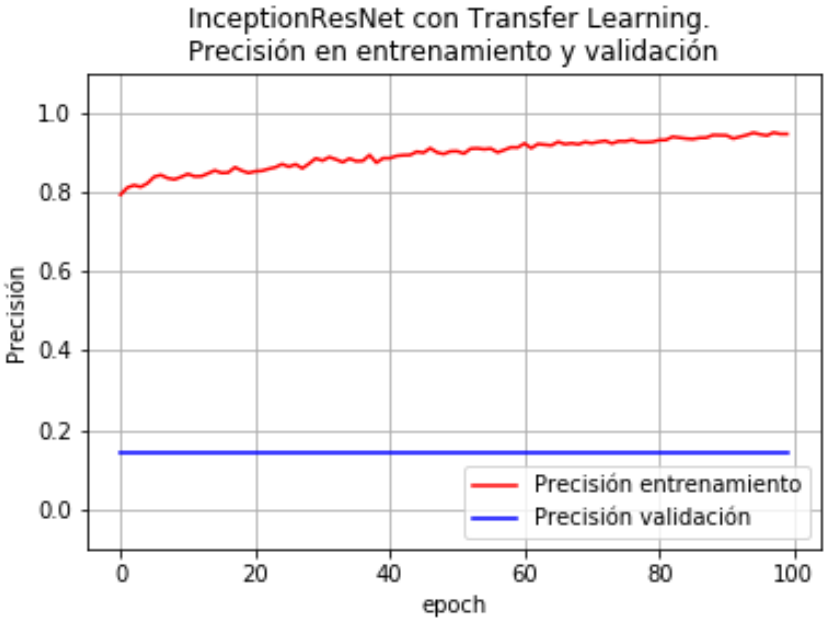


Ilustración 60 InceptionResNet con Transfer Learning. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

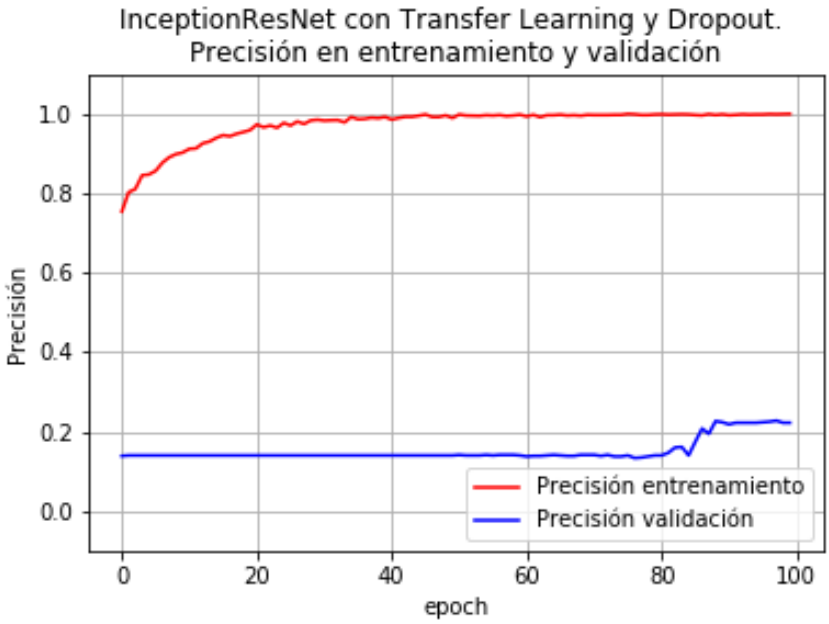


Ilustración 61 InceptionResNet con Transfer Learning y Dropout. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

Los resultados obtenidos son bastante malos: las redes alcanzan una precisión en entrenamiento cercana al 100%, mientras que en validación apenas superan el 20% en el mejor de los casos.

Como ya se indicó, esta técnica no siempre funciona y es muy dependiente de las imágenes utilizadas. En nuestro caso, las imágenes de nuestra base de datos son muy distintas a las imágenes utilizadas para entrenar las capas convolucionales de la red, lo que podría explicar estos resultados.

Ilustración 62 Diferencias entre las imágenes de ambos datasets. ImageNet (izqda.) y nuestro dataset (dcha.).

4.1.4. Pruebas preliminares con imágenes más grandes

Otra de las posibles soluciones al sobreajuste es utilizar más datos. En general, esto suele referirse a recopilar, preprocesar y etiquetar nuevas instancias y añadirlas

a nuestra base de datos. Esta solución presenta numerosas dificultades a esta altura del proyecto, sin embargo, sí que podemos utilizar más datos en forma de imágenes de mayor tamaño.

En todas las pruebas realizadas hasta este momento (excepto en la transferencia de conocimiento) se han utilizado las imágenes de 225 x 135 píxeles, no obstante, podemos usar las imágenes de tamaño 299 x 299 entrenando las arquitecturas completas (sin utilizar transferencia de conocimiento).

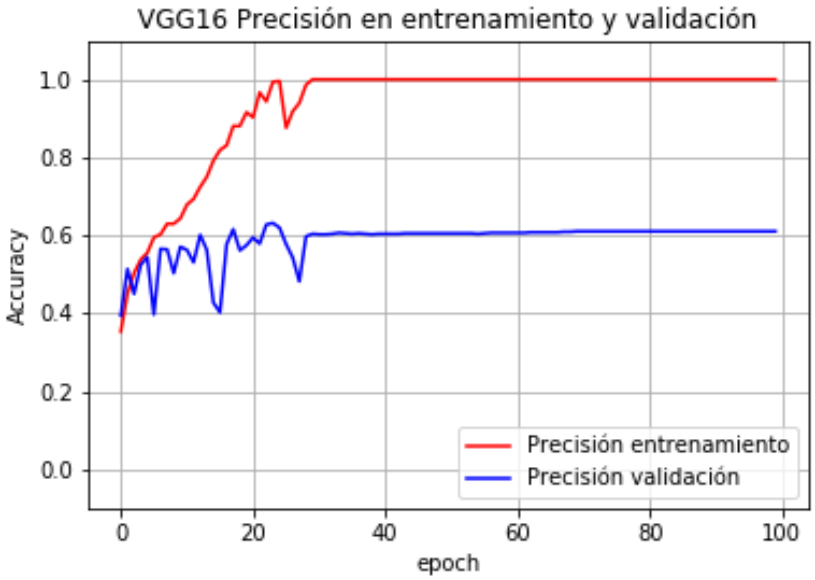


Ilustración 63 VGG-16 con imágenes 299x299. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

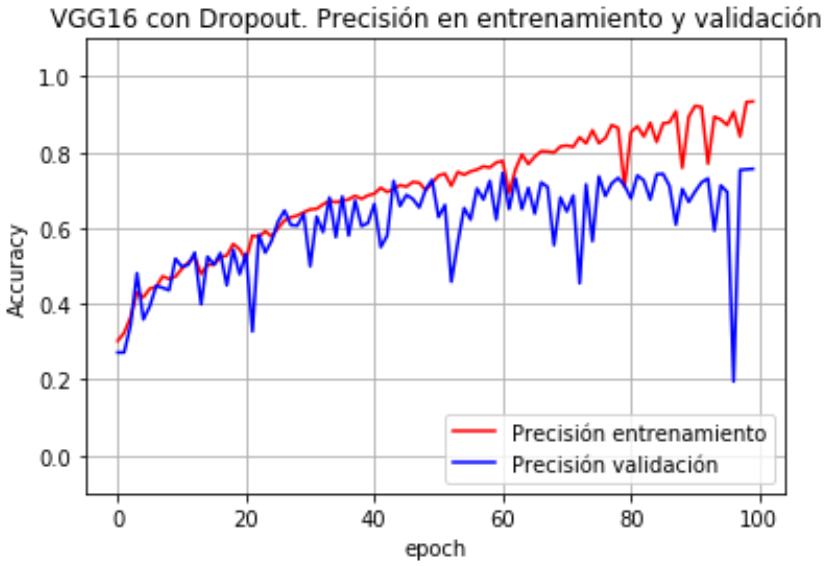


Ilustración 64 VGG-16 con imágenes 299x299 y Dropout. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

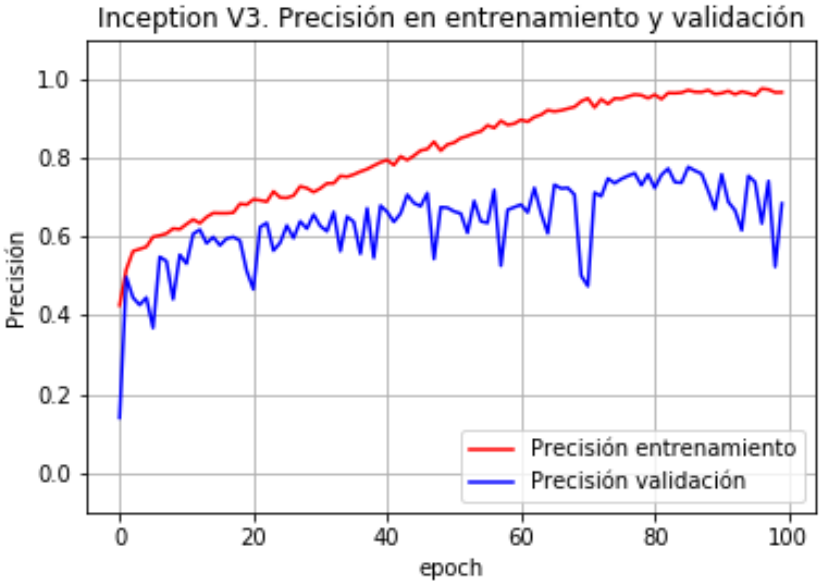


Ilustración 65 Inception V3 con imágenes 299x299. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

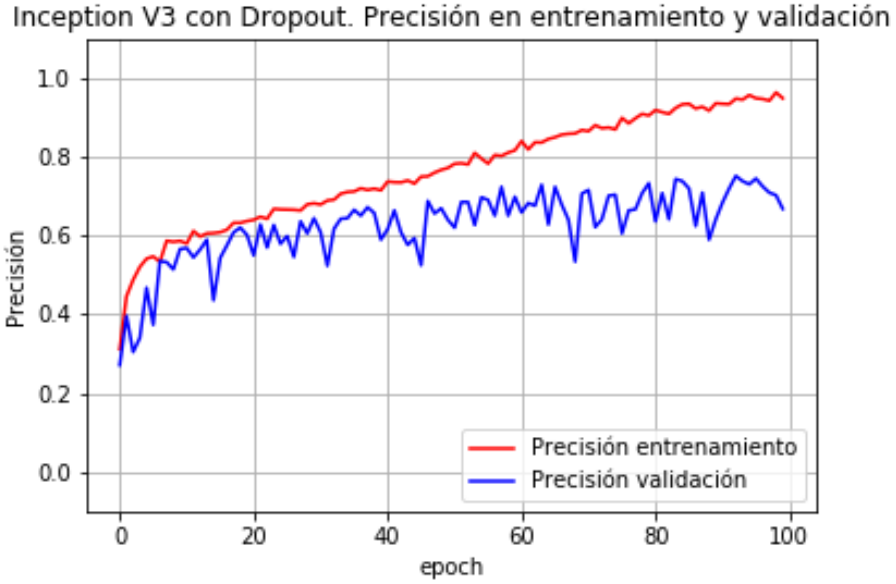


Ilustración 66 Inception V3 con imágenes 299x299 y Dropout. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

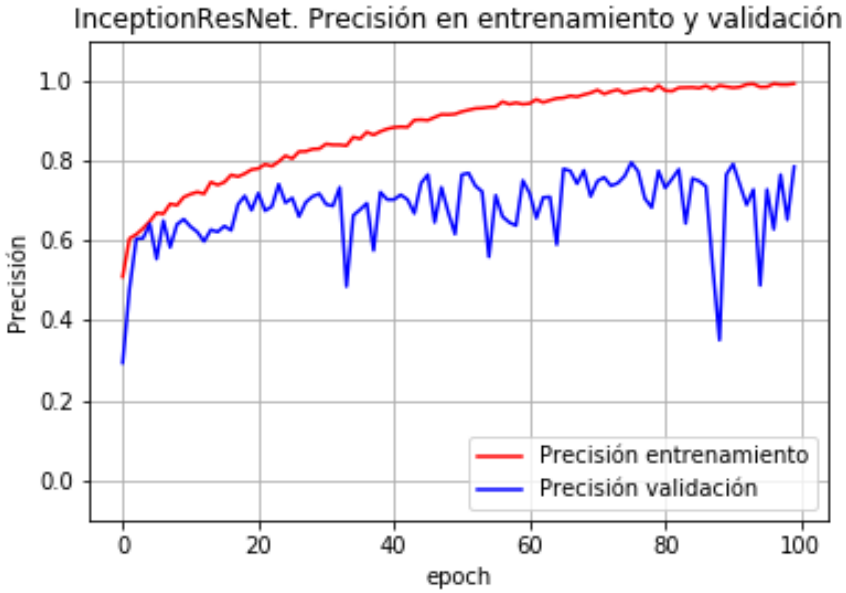


Ilustración 67 Inception ResNet con imágenes 299x299. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

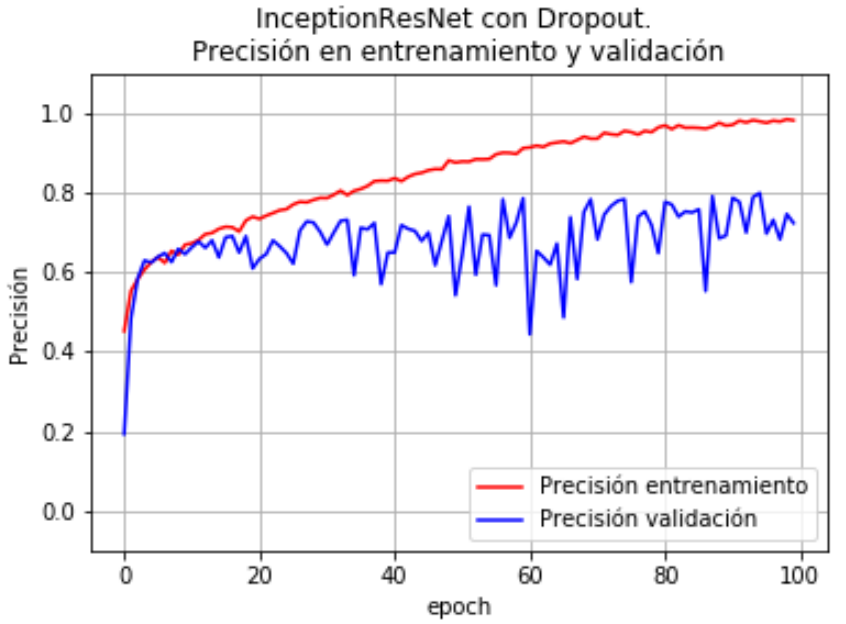


Ilustración 68 Inception ResNet con imágenes 299x299 y Dropout. Precisión en entrenamiento y validación. (Fuente: Elaboración propia).

Podemos observar que el uso de estas imágenes de mayor tamaño reduce (que no eliminan) el sobreajuste en prácticamente todas las arquitecturas y configuraciones utilizadas. También es importante notar que hay una ligera mejora en la precisión en validación de los modelos.

La última técnica para reducción del sobreajuste consiste en reducir la complejidad de la red. Esta técnica ya la hemos estado aplicando implícitamente al entrenar tres arquitecturas con distintas complejidades.

4.1.5. Selección de la arquitectura más prometedora

Una vez realizados estos primeros experimentos preliminares hemos de seleccionar uno de los modelos utilizados para realizar un ajuste de los hiperparámetros buscando la configuración que nos de los mejores resultados posibles.

A partir de los experimentos anteriores podemos resumir los siguientes resultados:

Las tres arquitecturas utilizadas presentan sobreajuste por lo que hemos de intentar reducirlo de alguna forma. De las técnicas probadas, las que mejores resultados han dado han sido:

- El uso de *Dropout* ha funcionado relativamente bien, reduciendo el sobreajuste en todas las pruebas realizadas.
- El aumento de datos y las imágenes de mayor tamaño también han dado buenos resultados.
- En cuanto a la reducción de la complejidad, todas las arquitecturas han alcanzado el estado de sobre entrenamiento, aunque en las redes más complejas (Inception V3 e InceptionResNet) este sobreajuste se produce antes y en mayor medida que en la red VGG-16.

Por tanto, para el modelo final se va a seleccionar la arquitectura VGG-16, con la cual utilizaremos las técnicas de Dropout, aumento de datos y las imágenes de tamaño 299 x 299.

4.2. Ajuste de hiperparámetros con búsqueda en cuadrícula

Una vez seleccionado este modelo y las técnicas que utilizaremos, el siguiente paso es realizar un ajuste de los hiperparámetros que nos den los mejores resultados en las imágenes de nuestra base de datos.

Como ya vimos en el capítulo 3, las CNNs presentan una gran cantidad de hiperparámetros a ajustar para obtener los mejores resultados; sería imposible con

los medios de que disponemos ajustar todos estos hiperparámetros. Por tanto, vamos a fijar varios de ellos y realizaremos una búsqueda en cuadrícula de los restantes.

Los parámetros que vamos a mantener fijos son:

- Los números de capas y de neuronas por capa, así como sus configuraciones (los tamaños de filtros de convolución, las funciones de activación, etc.) que vamos a utilizar son los correspondientes a la arquitectura VGG-16 original, descrita en [7].
- Para el *Dropout*, utilizaremos un valor de 0.5 aplicado a la última capa completamente conectada, la capa anterior a la capa de salida.
- Los valores para el aumento de datos que utilizaremos son los descritos en el apartado 4.1.2.

En la búsqueda en cuadrícula, por tanto, los hiperparámetros que buscaremos ajustar son: el **tamaño del lote** y el **número de *epochs***. Los valores utilizados para estos hiperparámetros se muestran en la siguiente tabla:

	Número de epochs	
Tamaño del lote	**200**	**300**
4	Experimento 1	Experimento 2
8	Experimento 3	Experimento 4

Tabla 5 Experimentos para búsqueda en cuadrícula (Fuente: Elaboración propia)

Se han elegido valores pequeños para el **tamaño del lote** debido principalmente a la limitación de recursos disponibles, tanto en tamaño de la base de datos como en memoria disponible para la computación. Eligiendo estos valores mantenemos pocas imágenes en memoria a la misma vez que conseguimos que la CNN actualice los pesos de sus parámetros muchas veces (566 veces para los experimentos 1 y 2, y 283 para los experimentos 3 y 4) en cada *epoch.*

Además de las gráficas de precisión (*accuracy*) que hemos estado utilizando en las pruebas preliminares, también estudiaremos las gráficas de la evolución del error o pérdida (*loss*) cometida, así como la matriz de confusión y la métrica F1 obtenidas para el conjunto de validación tras el entrenamiento completo de la red.

A continuación se presentan los resultados obtenidos en cada uno de estos experimentos.

4.2.1. Experimento 1: 200 epochs y tamaño de lote 4

En el primer experimento se ha entrenado la red durante 200 *epochs,* utilizando un tamaño de lote de 4. La red ha tardado 5.33 horas en entrenarse, obteniendo un valor para el F1-Score de: 0.77695.

En las siguientes gráficas podemos ver el resto de métricas estudiadas:

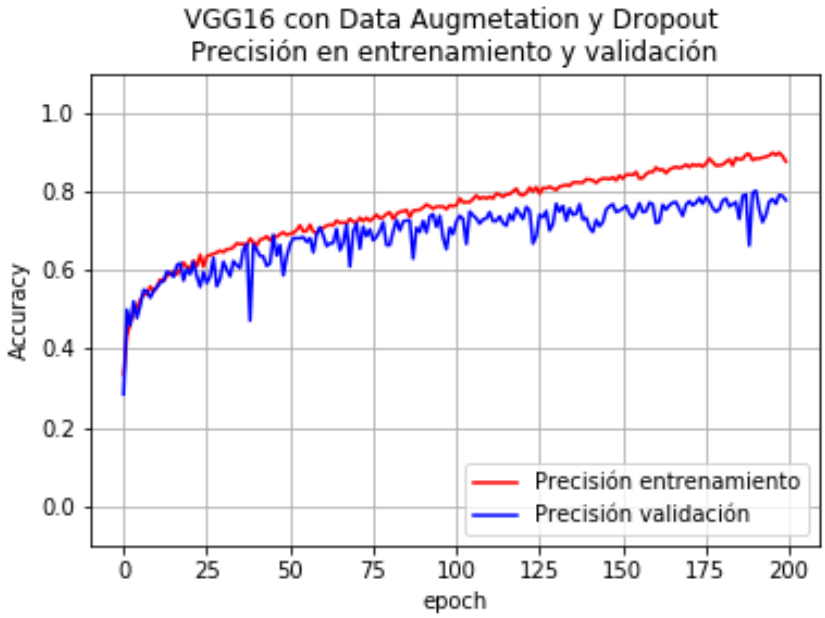


Ilustración 69 Experimento 1. Precisión en entrenamiento y validación por epoch. (Fuente: Elaboración propia)

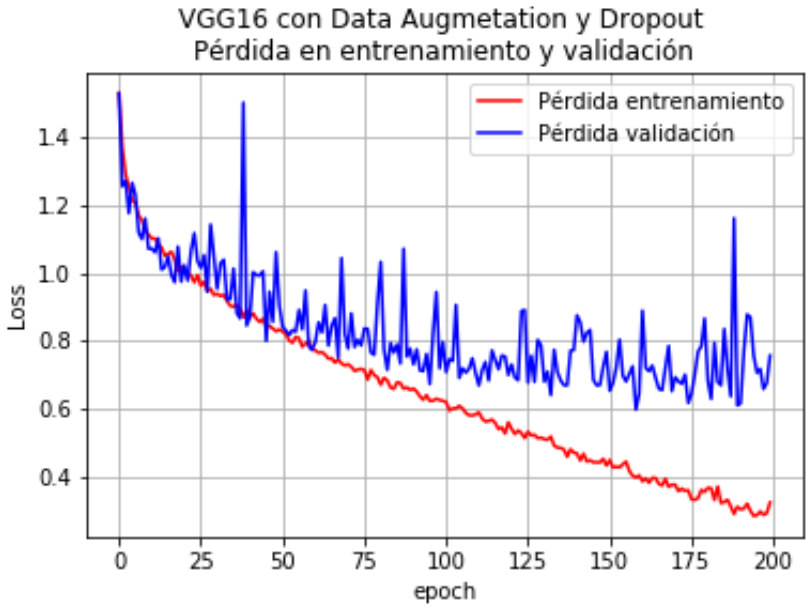


Ilustración 70 Experimento 1. Pérdida en entrenamiento y validación por epoch. (Fuente: Elaboración propia)

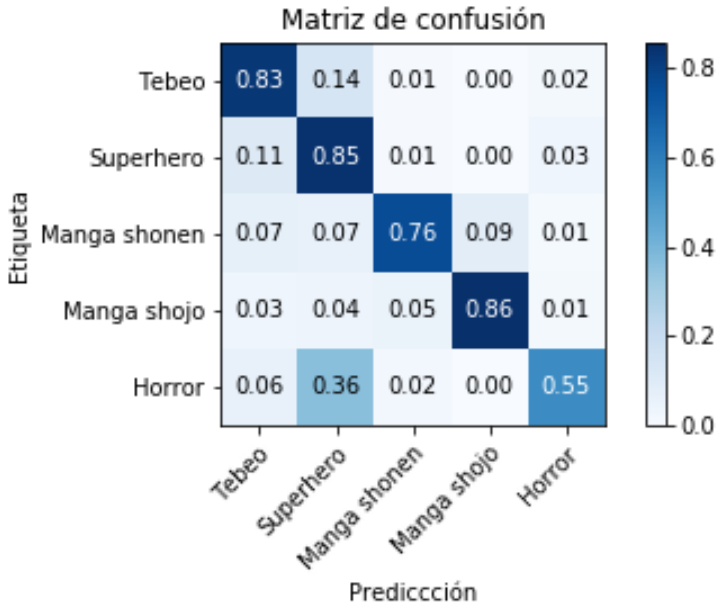


Ilustración 71 Experimento 1. Matriz de confusión. (Fuente: Elaboración propia)

En la gráfica de precisión podemos ver que ambas curvas se mantienen muy cerca una de la otra a lo largo de todo el proceso de entrenamiento, alcanzando una separación máxima del 10%, aproximadamente, con una precisión máxima en validación del 80%.

Si observamos la gráfica de pérdidas vemos que la pérdida en entrenamiento decrece a lo largo de todo el proceso, mientras que la pérdida en validación parece

que deja de decrecer (o lo hace mucho más lentamente) en la segunda mitad del proceso. Además, tanto en la gráfica de precisión como en la de pérdida podemos observar que la curva en validación es muy ruidosa (un ruido más pronunciado en la gráfica de pérdida), mientras que las curvas en entrenamiento son más estables. Este ruido se debe probablemente al pequeño valor elegido para el tamaño del lote, que aumenta la variabilidad del modelo.

En cuanto a la matriz de confusión, vemos que los valores más oscuros (mejores valores de clasificación) están en la diagonal principal, obteniendo un porcentaje de acierto superior al 80% en 3 de las categorías (*Tebeo, Superhero y Manga Shojo*) y del 76% en la categoría *Manga Shonen*. Los peores resultados los obtenemos en la categoría *Horror*, en la que un 36% de las instancias se han clasificado como *Superhero*.

4.2.2. Experimento 2: 300 epochs y tamaño de lote 4

En el segundo experimento se ha entrenado la red durante 100 *epochs* más, para un total de 300 *epochs,* utilizando igualmente un tamaño de lote de 4. La red ha tardado unas 8 horas en entrenarse, obteniendo un valor para el F1-Score de: 0.80083.

Las gráficas con el resto de métricas estudiadas:

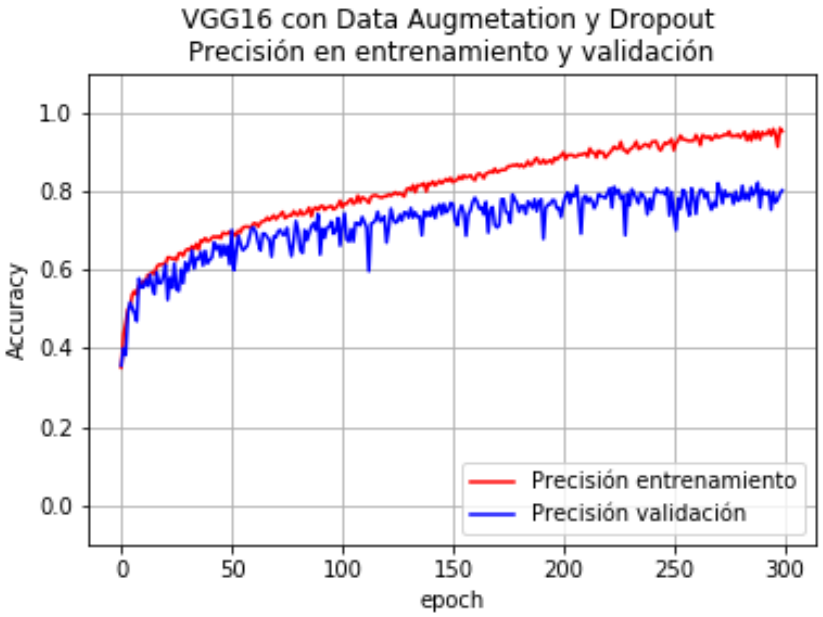


Ilustración 72 Experimento 2. Precisión en entrenamiento y validación por epoch. (Fuente: Elaboración propia)

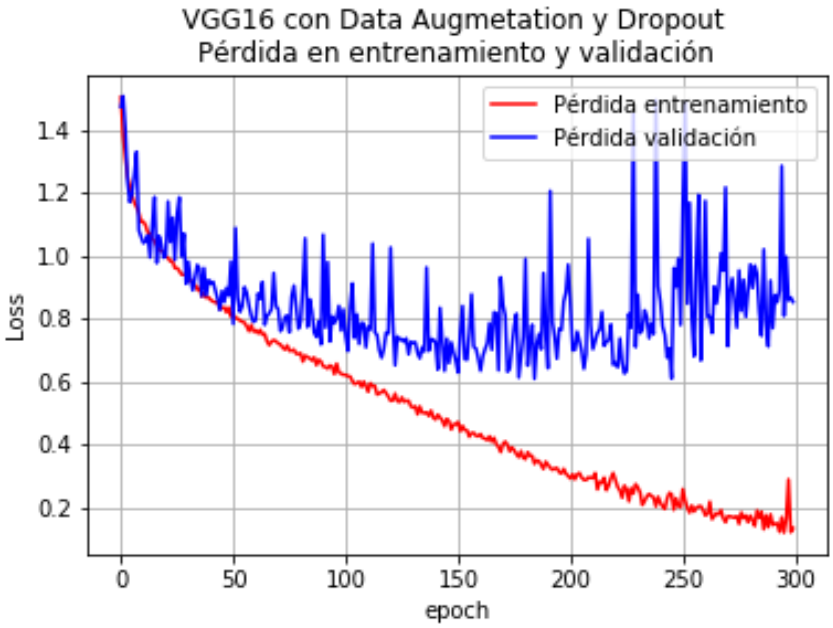


Ilustración 73 Experimento 2. Pérdida en entrenamiento y validación por epoch. (Fuente: Elaboración propia)

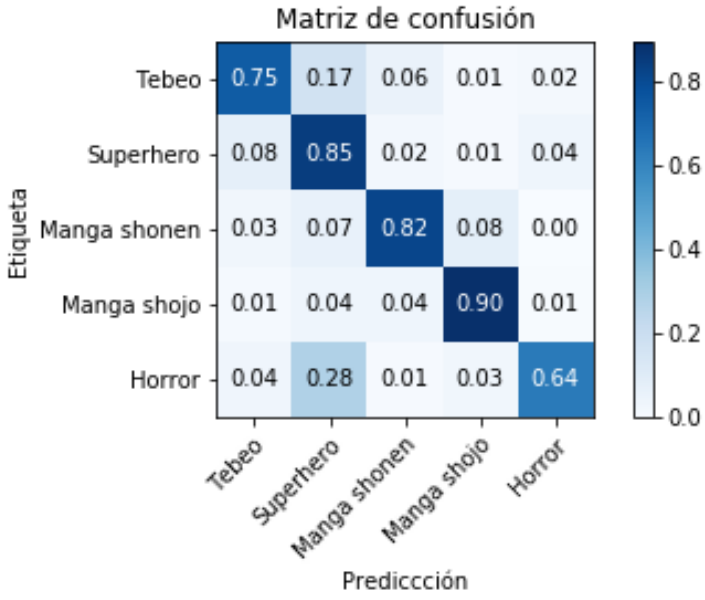


Ilustración 74 Experimento 2. Matriz de confusión. (Fuente: Elaboración propia)

Los resultados obtenidos para la gráfica de precisión son similares a los del experimento anterior, algo esperable, en los que la curva de precisión en entrenamiento continúa mejorando lentamente, mientras que la de validación se mantiene estable en torno al 80%.

En la gráfica de pérdidas, sin embargo, se observan mayores diferencias. Si obviamos el ruido en la curva de validación y nos fijamos en la tendencia de la curva,

se aprecia que, a partir del segundo tercio de los *epochs*, esta pérdida vuelve a aumentar, obteniendo peores resultados. Se trata de un caso en el que está aumentando el sobreajuste del modelo y en el que probablemente no obtengamos mejores resultados con un mayor número de *epochs.*

En cuanto a la matriz de confusión, vemos que los resultados son algo mejores a los del experimento anterior, mejorándose en casi todas las categorías. Al igual que en el primer experimento, los mayores errores se comenten al clasificar como *Superhero* instancias que pertenecen a *Horror* (28% de las instancias de *Horror*) y a *Tebeo* (17%).

4.2.3. Experimento 3: 200 epoch y tamaño de lote 8

En el tercer experimento se ha entrenado la red durante un total de 200 *epochs,* utilizándose en este caso un tamaño de lote de 8. La red ha tardado unas 4.8 horas en entrenarse, obteniendo un valor para el F1-Score de: 0.72027.

Las gráficas con el resto de métricas estudiadas se presentan a continuación:

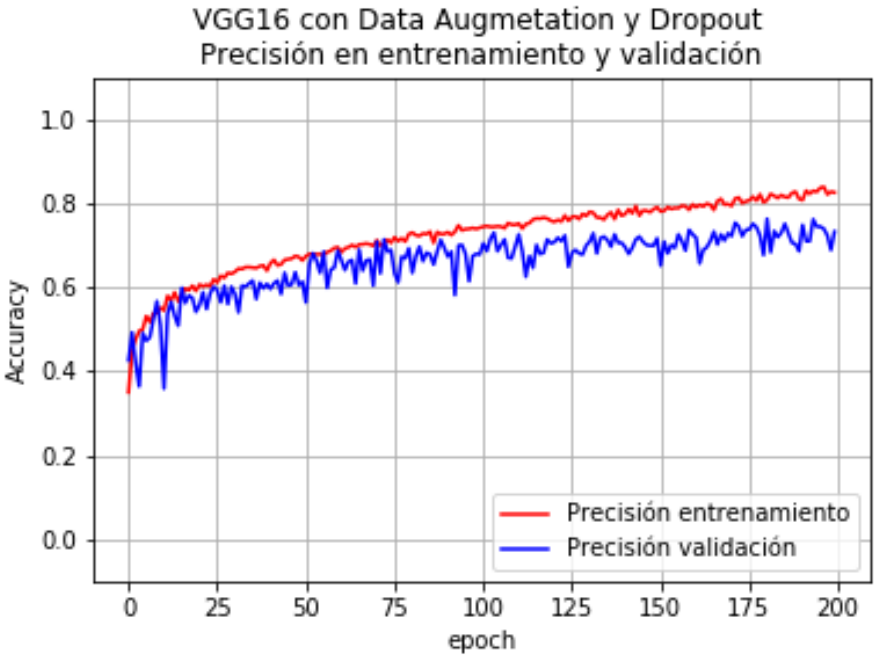


Ilustración 75 Experimento 3. Precisión en entrenamiento y validación, por epoch. (Fuente: Elaboración propia)

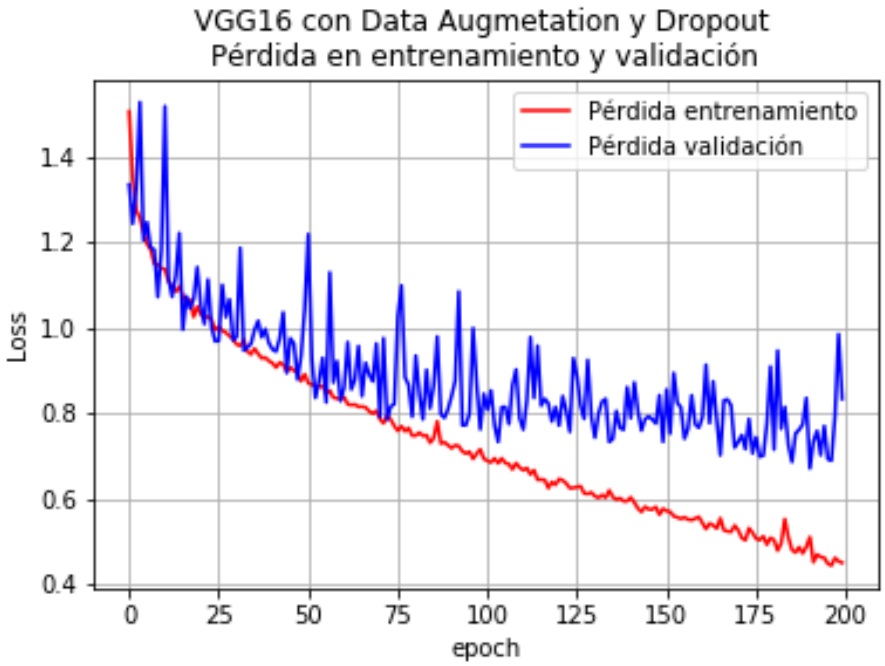


Ilustración 76 Experimento 3. Pérdida en entrenamiento y validación, por epoch. (Fuente: Elaboración propia)

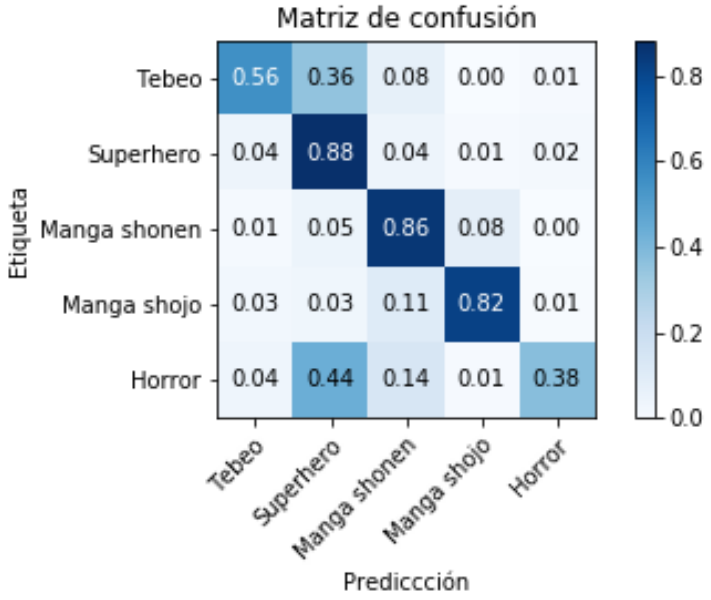


Ilustración 77 Experimento 3. Matriz de confusión. (Fuente: Elaboración propia)

Las topologías de las gráficas de precisión y pérdida para este experimento son muy similares a las de las gráficas del Experimento 1. Los valores de entrenamiento mejoran lenta pero constantemente a lo largo del proceso de aprendizaje, mientras que los valores de validación mejoran más lentamente, conteniendo mucho más ruido que la curva de entrenamiento. En el caso de la pérdida en validación, además, en la segunda mitad del proceso, la curva se separa bastante de la de entrenamiento.

En cuanto a la matriz de confusión, volvemos a obtener muy buenos resultados, de entre el 80 y el 90% de acierto, en 3 categorías (*Superhero, Manga shonen y Manga shojo*), mientras que los resultados de las otras 2 categorías empeoran significativamente (un 56% de acierto en la clase *Tebeo* y solo un 38% en la clase *Horror*).

4.2.4. Experimento 4: 300 epoch y tamaño de lote 8

En el cuarto y último experimento se ha entrenado la red durante un total de 300 *epochs,* utilizándose un tamaño de lote de 8. La red ha tardado unas 8 horas en entrenarse, obteniendo un valor para el F1-Score de: 0.79055.

Las gráficas con las métricas estudiadas se presentan a continuación:

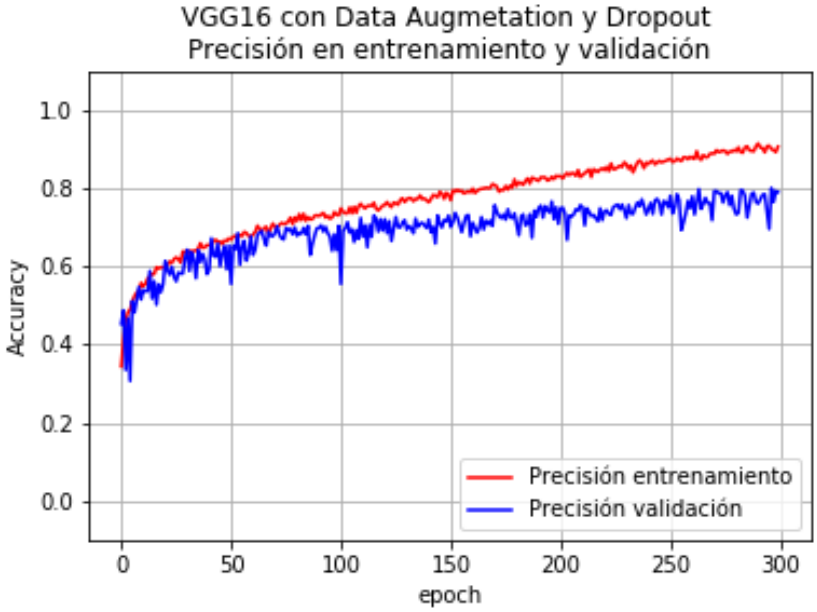


Ilustración 78 Experimento 4. Precisión en entrenamiento y en validación, por epoch. (Fuente: Elaboración propia)

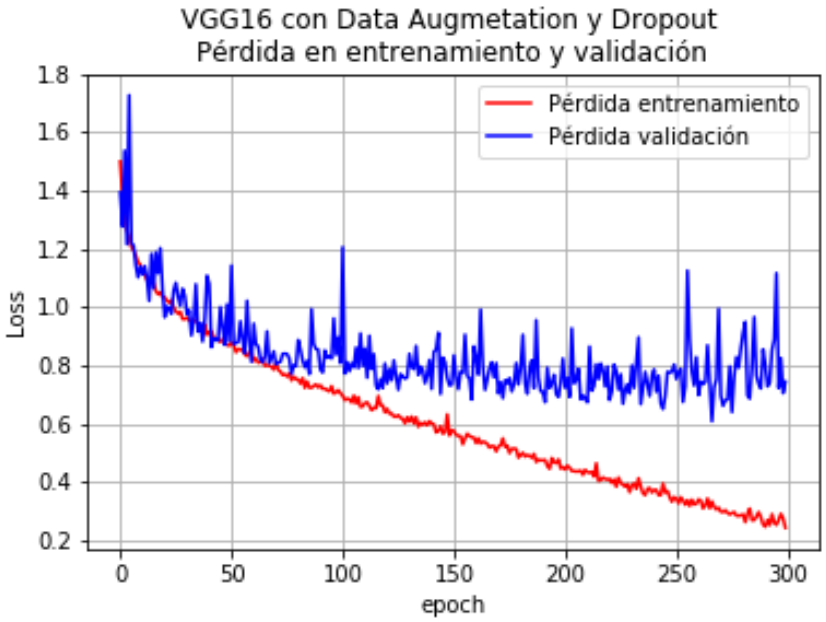


Ilustración 79 Experimento 4. Pérdida en entrenamiento y en validación, por epoch. (Fuente: Elaboración propia)

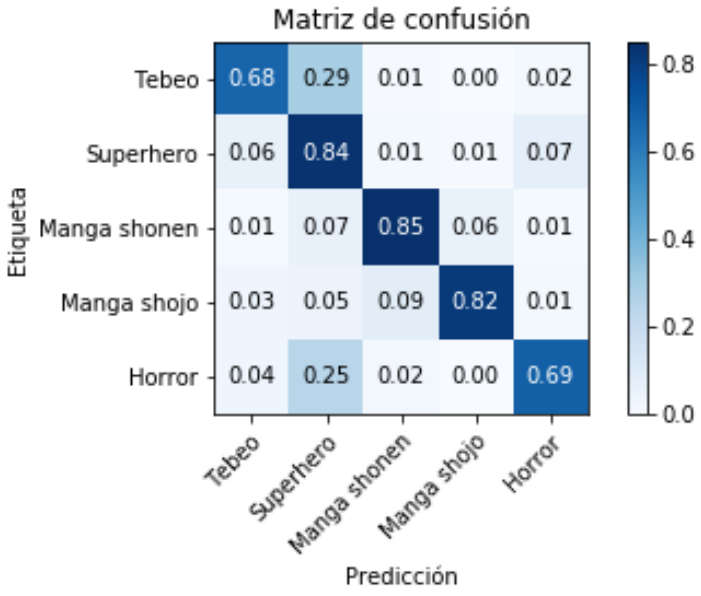


Ilustración 80 Experimento 4. Matriz de confusión. (Fuente: Elaboración propia)

A igual que ocurría en el segundo experimento, los resultados obtenidos son muy similares a los del experimento anterior, pero con más información, ya que hemos entrenado la red durante más *epochs*.

Para la gráfica de precisión, vemos que ambas curvas se mantienen bastante cercanas, aunque a mediados del entrenamiento se separan, alcanzando una

separación de un 10% aproximadamente, y en la que se obtiene una precisión para validación de un 80%.

En la gráfica de pérdida vemos que la curva de entrenamiento decrece continuamente, mientas que la de validación parece estancarse (obviando el ruido de la curva) en un valor entre 0.6 y 0.8.

En cuanto a la matriz de confusión, vemos que en las clases *Superhero, Manga shonen y Manga shojo* se obtienen buenos resultados (en torno al 85% de acierto), mientras que en las otras dos clases (*Tebeo y Horror*) los resultados son peores (69%, aproximadamente), aunque hay una notable mejoría con respecto al experimento anterior.

4.2.5. Resumen de los resultados de los experimentos

Una vez realizada esta búsqueda en cuadrícula, vamos a seleccionar uno de los modelos para realizar una predicción final sobre el subconjunto de test y evaluar los resultados obtenidos.

	Experimento 1	Experimento 2	Experimento 3	Experimento 4
F1-Score	0.77695	0.80083	0.72027	0.79055

Tabla 6 Resumen del F1-Score de los 4 experimentos (Fuente: Elaboración propia)

Basándonos en los resultados ya analizados de los 4 experimentos y en la tabla anterior en la que hemos reunido los valores del *F1-Score* de todos estos experimentos, observamos que los mejores resultados se obtienen con los modelos de los experimentos 2 y 4.

Si nos fijamos solamente en los valores del *F1-Score,* el modelo del experimento 2 es mejor (un 0.01) que el del experimento 4. Sin embargo, hemos visto que las curvas de precisión y pérdida en validación contienen una alta variabilidad en forma de ruido. Observando las tendencias de las curvas de pérdidas en validación para ambos experimentos, vemos que en el Experimento 2 presentaba una tendencia a empeorar (aumentar el valor de pérdida), mientras que en el Experimento 4 se mantenía estable dentro del ya mencionado ruido.

Debido a esto, es probable que el modelo del Experimento 4 sea mejor o más robusto que el del Experimento 2, por tanto, será este modelo el que utilizaremos para realizar la predicción final

4.3. Predicción del modelo

Finalmente, tal y como se ha explicado en el punto anterior, vamos a utilizar el modelo del Experimento 4 (tamaño de lote 8 y 300 epochs) para predecir las clases de las instancias del subconjunto de prueba o test y evaluaremos el modelo en función de estos resultados. Para esta evaluación vamos a utilizar la matriz de confusión y el *F1-Score*.

El valor obtenido para el *F1-Score* es 0.78847 y la matriz de confusión la vemos en la siguiente imagen.

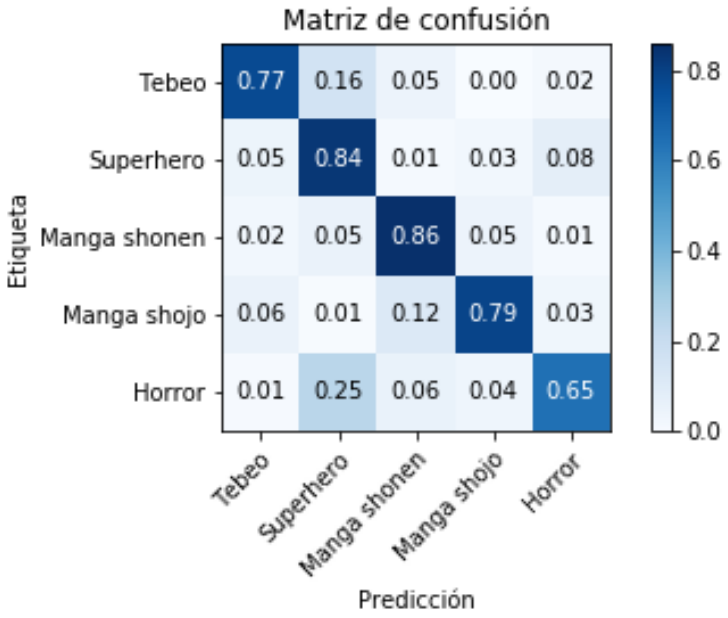


Ilustración 81 Matriz de confusión para las predicciones del modelo final. (Fuente: Elaboración propia)

Los resultados obtenidos son muy similares, aunque un poco inferiores, a los de la validación del modelo, vistos en el apartado 4.2.4. El valor del *F1-Score* solamente difiere del de validación en 0.02, mientras que en la matriz de confusión los valores de la diagonal principal son más homogéneos que en validación. Para las clases *Superhero, Manga shonen y Manga shojo* los resultados son algo peores, pero en las clases *Tebeo y Horror*, la clasificación mejora considerablemente. Al igual que en los otros modelos, los mayores errores se cometen al clasificar instancias de las clases *Horror y Tebeo* como *Superhero.*

7. Conclusiones

En este último capítulo se presentan las concusiones extraídas de los distintos experimentos realizados a lo largo de todo el proceso y se proponen algunas ideas para futuros trabajos.

A lo largo del proceso, en todas las pruebas realizadas, hemos comprobado que los modelos que hemos utilizado alcanzaban, en mayor o menor medida, el estado de sobreajuste. Aunque hemos aplicado algunas medidas para prevenirlo (*dropout*, aumento de datos, etc.), solamente hemos conseguido reducirlo, pero no eliminarlo.

El principal motivo de que alcancemos este estado es la escasa cantidad de imágenes disponibles en la base de datos, unas 3700 imágenes. Además, tras dividir el conjunto de datos en los subconjuntos de entrenamiento, validación y test, las CNNs utilizadas solamente disponían de unas 2200 imágenes de las que extraer las características de los distintos cómics.

Algunos de los conjuntos de datos más utilizados y conocidos para clasificación de imágenes disponen de un número mucho mayor de instancias. Por ejemplo, el ya mencionado *ImageNet* se compone de más de 14 millones de imágenes. Incluso un *dataset* tan simple como MNIST contiene 70000 imágenes.

Otro de los problemas de los distintos modelos lo hemos observado en las matrices de confusión. En ellas hemos visto que para las clases *Superhero, Manga shonen y Manga shojo*, en general, obtenemos buenos porcentajes de acierto, de entre el 80 y el 90%. Sin embargo, para las clases *Tebeo y Horror,* muchas de las instancias se clasifican incorrectamente como *Superhero*.

Es posible que las imágenes de estas tres clases posean características similares, ya que todas ellas pertenecen a la superclase de *cómic occidental.* En particular, muchos de los cómics de la clase *Horror* están dibujados y editados por las mismas compañías que producen los cómics de la clase *Superhero*, aunque ambos tipos de cómics cuenten historias de distinta temática.

7.1. Propuestas de trabajo futuro

Tal y como vimos en el apartado 2.1, el Proceso de Descubrimiento de Conocimiento en Bases de Datos es un proceso iterativo e interactivo. Por tanto, es habitual que, tras obtener los resultados de los primeros modelos, se vuelva a etapas

anteriores y se reevalúen las decisiones tomadas. Las distintas propuestas que se presentan a continuación van en esta línea, en particular, en volver a las primeras etapas del proceso, en las que recabamos y estudiamos los datos.

La primera propuesta es la más obvia de todas tras ver los resultados obtenidos. La propuesta es simplemente conseguir más imágenes para nuestra base de datos. Esto presenta, no obstante, algunas complicaciones.

Las imágenes que estamos estudiando pertenecen a una temática muy concreta que, además, están sujetas a derechos de autor. En este proyecto hemos utilizado la API del buscador Bing de Microsoft para recopilar las imágenes. Sin embargo, el número de imágenes disponibles a través de esta herramienta es muy limitado.

Existen múltiples herramientas para leer cómics en internet, tanto de pago (casi todas las editoriales o productoras tienen servicios de subscripción para leer cómics en sus webs), como gratuitos (aunque la gran mayoría de ellos son ilegales). Es probable que se puedan utilizar obtener imágenes de estos servicios utilizando algún tipo de araña (*web crawler*) o script.

Otra posible forma de conseguir más datos consiste en estudiar las imágenes para realizar un aumento de datos “manual”. Por ejemplo, la gran mayoría de las imágenes de la base de datos son de altas dimensiones en las que se pierde mucha información tras realizar el preprocesamiento (recortado + redimensionado). Muchas de estas imágenes están divididas en viñetas, por lo que, tal vez, se podría realizar un recortado manual de estas viñetas para conseguir más imágenes. Por ejemplo, en la siguiente imagen se han marcado con rectángulos rojos las cuatro viñetas del tebeo. Se podrían obtener de esta imagen 5 instancias para la base de datos, la imagen original más otra imagen por cada viñeta.

Ilustración 82 Propuesta de aumento de datos manual (Fuente: Elaboración propia)

La última propuesta consiste en volver a estudiar las decisiones tomadas en el apartado 4.2, con respecto a las clases de las imágenes. Como ya hemos visto en las matrices de confusión de los experimentos, parece existir cierto solapamiento entre algunas clases, en particular, entre *Horror* y *Superhero.* Habría que estudiar en más detalle, tal vez consultar con algún experto en ilustraciones o en cómics, para que nos ayuden a etiquetar correctamente las imágenes de nuestra base de datos.

Una posible variación de esta propuesta, considerando además el pequeño número de imágenes disponibles en el conjunto de datos actual, sería la de simplificar el número de clases del problema. Como vimos en el apartado 4.2, las 5 clases que aquí hemos estudiado pertenecen a dos super clases: *cómics occidentales y cómics orientales o mangas.* Es posible que un modelo de clasificación binario, que diferencie entre estas dos clases nos proporcione información útil, incluso que nos lleve a un sistema en dos etapas: un modelo que distinga entre cómics occidentales y mangas y, una vez hecha esta clasificación, otros dos modelos que clasifiquen entre *Tebeo, Superhero y Horror* en el primer caso y entre *Manga shonen y Manga shojo* en el segundo.

Bibliografía

[1] I. Goodfellow, Y. Bengio y A.Courville, «Deep Learning», MIT Publisher, 2016

[2] J. Hernández Orallo, M.J. Ramírez Quintana y C. Ferri Ramírez. «Introducción a la Minería de Datos», Pearson Prentice Hall, 2005

[3] I. H. Witten, E. Frank y M. A. Hall, «Data Mining: Practical Machine Learning Tools and Techniques», Morgan Kaufmann, 2011

[4] Y. LeCun et al., «Gradient-Based Learning Applied to Document Recognition», *Proceedings of the IEEE 86*, 2278-2324, 1998

[5] N. Srivastava et al., «Dropout: A Simple Way to Prevent Neural Networks from Overfitting», *Journal of Machine Learning,* pp.1929-1958, 2014

[6] A. Krizhevsky, et al., «ImageNet Classification with Deep Convolutional Neural Networks», *Advances in neural information processing systems 25,* 2012

[7] K. Simonyan y A. Zisserman, «Very Deep Convolutional Networks for Large-Scale Image Recognition», *ICLR*, 2014

[8] C. Szegedy, W. Liu, Y. Jia et al., «Going Deeper with Convolutions», *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recogni.,* 2014

[9] C. Szegedy, V. Vanhoucke, S. Ioffe et al., «Rethinking the Inception Architecture for Computer Vision», 2015

[10] K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition», *CoRR,* 2015

[11] C. Szegedy, S. Ioffe, V. Vanhoucke y A. Alemi, «Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning», 2016

[12] J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.

[13] N. S. Altman. «An introduction to kernel and nearest-neighbor nonparametric regression.». *The American Statistician.* **46** 175–185. 1992

[14] Y. Freund, R. E. Schapire. «Large margin classification using the perceptron algorithm.». *Machine Learning.* 277–296. 1999

[15] J. Ahmad, K. Muhammad, S. W. Baik, «*Data augmentation-assisted deep Learning of Hand-Drawn Partially Colored Sketches for Visual Research*». 2017

[16] J. Hale, «Deep Learning Framework Power Scores 2018» [En línea]. Disponible: https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a, 2018

[17] «TensorFlow documentation», [En línea]. Disponible: http://www.tensorflow.org/.

[18] «Keras documentation», [En línea]. Disponible: http://www.keras.io/.

[19] «Caffe documentation», [En línea]. Disponible: https://caffe.berkeleyvision.org/

[20] «Microsoft Cognitive Toolkit documentation», [En línea]. Disponible: https://docs.microsoft.com/en-us/cognitive-toolkit/

[21] «Pytorch documentation», [En línea]. Disponible: https://pytorch.org/

[22] «MXNET documentation», [En línea]. Disponible: https://mxnet.apache.org/

[23] «Chainer Documentation», [En línea]. Disponible: https://chainer.org/

[24] «Cuda Toolkit Documentation», [En línea]. Disponible: https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html.

[25] «Anaconda documentation», [En línea]. Disponible https://www.anaconda.com/.

[26] «Google Colab Documentation», [En línea]. Disponible: https://colab.research.google.com/.

[27] L. G. Pérez Cordón. «Minería de datos». Grado en Ingeniería Informática, Universidad de Jaén. 2017/2018.

[28] A. Ng. «Deep Learning». Deep Learning Specialization, Coursera. 2018.

[29] «FaceApp», [En línea]. Disponible https://www.faceapp.com/.

[30] «AlphaGo», [En línea]. Disponible https://www.deepmind.com/research/alphago/.

Anexo I. Instalación de un entorno local para Deep Learning

En este anexo se presentan los programas e instrucciones necesarias para que un usuario pueda recrear el sistema para trabajar con tecnologías Deep Learning presentado en este proyecto o uno similar. Los elementos incluidos en este anexo son:

1. Instalación del entorno de desarrollo para Deep Learning.
2. Instalación de Cuda y CuDNN para aceleración del entrenamiento de las redes neuronales.
3. Configuración del entorno con los paquetes o bibliotecas de Python necesarias para el funcionamiento del sistema.

El primer paso necesario para el funcionamiento del sistema es la instalación y configuración de un entorno para el desarrollo en Deep Learning. Para este proyecto el entorno utilizado ha sido la suite Anaconda en Windows 7, para el desarrollo con Python. Sobre este entorno se han utilizado las bibliotecas de Deep Learning TensorFlow y Keras, además de CUDA y la biblioteca para redes profundas de Cuda (CuDNN), para poder aprovechar la potencia de la tarjeta gráfica (GPU) en la computación necesaria para el sistema.

1. Instalación del entorno de desarrollo

A continuación se presentan las instrucciones para la instalación de la suite Anaconda.

1.1. Descarga de Anaconda

Anaconda es un entorno de desarrollo en Python orientado a uso científico, gratuito y fácil de utilizar. El primer paso consiste en descargar la última versión de Anaconda (necesitamos la versión con Python 3.7) adecuada para nuestro sistema operativo, desde la página web oficial de Anaconda: https://www.anaconda.com/download/.

1.2. Instalación de Anaconda

Una vez descargado el ejecutable, hemos de instalar el entorno. El proceso es muy sencillo y rápido, en el que simplemente tenemos que seguir las instrucciones del

asistente. En el proceso de instalación se instalará, además del entorno de desarrollo, la versión adecuada de Python, así como algunos paquetes básicos.

Una vez el proceso haya finalizado lanzamos Anaconda (si no se ha lanzado automáticamente) e instalaremos las aplicaciones *Jupyter Notebook* y *Spider*, en el caso de que no se hayan instalado por defecto.

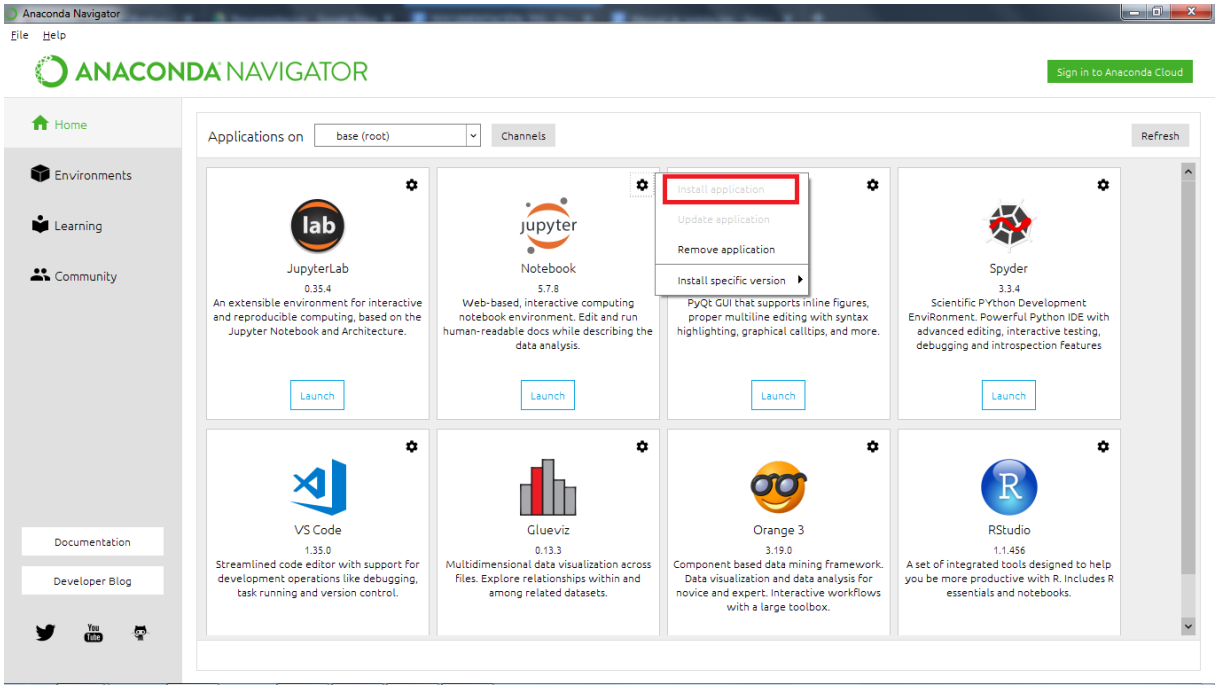


Ilustración 83 Pantalla inicial de Anaconda, con detalle en rojo para instalación de aplicaciones (Fuente: Elaboración propia a partir de Anaconda Navigator).

1.3. Actualización de Anaconda

Cuando hayamos instalado la suite Anaconda y las aplicaciones *Jupyter Notebook* y *Spider*, es conveniente actualizar el entorno y todos sus componentes a las últimas versiones disponibles. Para ello, en el *Anaconda Prompt* (consola de comandos instalada con el entorno Anaconda) ejecutaremos los siguientes comandos:

```
conda update conda
conda update --all
```

2. Instalación y configuración de Cuda y CuDNN

2.1. Actualización de los drivers de la tarjeta gráfica NVIDIA

En primer lugar, hemos de actualizar los controladores (*drivers* en inglés) del hardware gráfico NVIDIA a la versión más reciente. Para ellos descargaremos desde la página oficial de NVIDIA la última versión de los controladores de nuestra tarjeta.

A continuación instalaremos esta versión de los *drivers* siguiendo las instrucciones del asistente.

Una vez actualizados, hemos de comprobar cuál es la versión de estos controladores que tenemos funcionando en el sistema. Para ello abriremos el *Panel de control de NVIDIA* y en la opción *Información del Sistema* buscaremos y tomaremos nota de la versión del controlador.

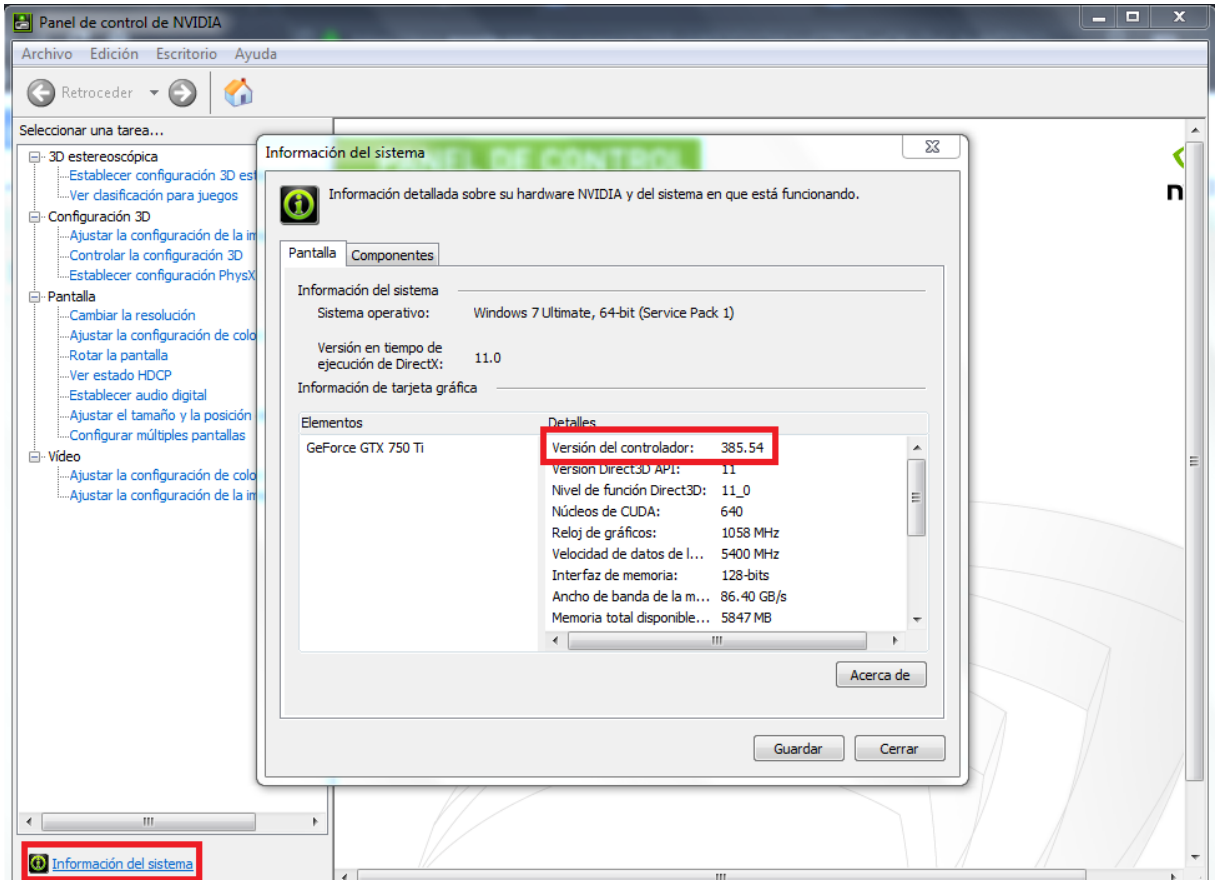


Ilustración 84 Detalle de la versión del controlador de gráficos NVIDIA (Fuente: Elaboración propia).

2.2. Descarga del CUDA Toolkit 9.0

Con los drivers de la tarjeta gráfica actualizados descargaremos la versión 9.0 del CUDA Toolkit[4] adecuada para nuestro Sistema Operativo desde la página oficial. Descargaremos también todos los parches que acompañen a la instalación base del Toolkit.

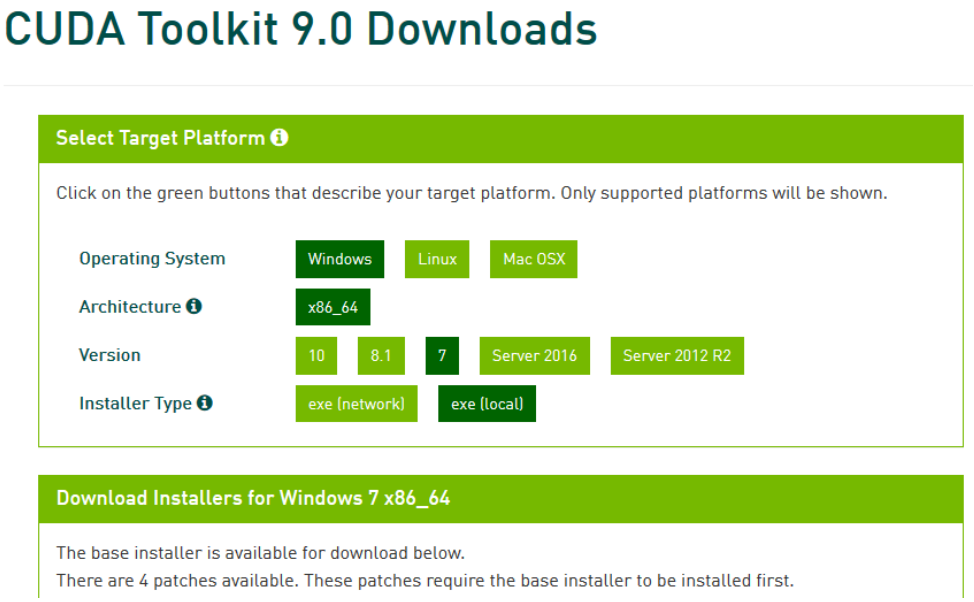


Ilustración 85 Página de descarga del CUDA Toolkit 9.0 (Fuente: NVIDIA).

2.3. Instalación del CUDA Toolkit 9.0

A continuación instalaremos la base del CUDA Toolkit y todos los parches descargados, siguiendo las instrucciones de los asistentes.

2.4. Descarga de CuDNN

También descargaremos la última versión disponible de CuDNN desde la página oficial. Descargaremos la versión adecuada para nuestro Sistema Operativo y la versión instalada de CUDA.

Para poder descargar nos pedirán que nos registremos con un correo electrónico y que accedamos a los términos de uso.

[4] En el momento de redactar este proyecto las versiones 9.1 y 9.2 del CUDA Toolkit no son compatibles con TensorFlow. Es posible que cuando se lea este documento estas compatibilidades hayan cambiado. Consultar la documentación oficial en caso de duda.

Ilustración 86 Página de descarga de CuDNN (Fuente: NVIDIA).

2.5. Instalación de CuDNN

Para instalar CuDNN hemos de descomprimir el fichero .zip descargado en la carpeta C, de la siguiente manera:

```
C:\cudnn-9.0-windows7-x64-v7.4.2
```

A continuación hemos de añadir CuDNN al *Path* del sistema. Para ello hemos de seguir los pasos:

1. Abrir el diálogo de `Ejecutar` (Win + R) y ejecutar el comando **`sysdm.cpl`**.
2. En las propiedades del sistema elegir la pestaña **Opciones avanzadas**.
3. Seleccionar **Variables de entorno**.
4. Añadir la siguiente ruta al Entorno:

```
C:\cudnn-9.0-windows7-x64-v7.4.2\cuda\bin
```

3. Configuración del entorno en Anaconda

Una vez realizados los pasos anteriores, ya podemos instalar las librerías de Deep Learning, pero antes crearemos un entorno de Anaconda para que los cambios no afecten a la instalación raíz o base de Anaconda.

3.1. Creación de un entorno en Anaconda

En el *Prompt Anaconda* ejecutaremos los siguientes comandos:

1. Para crear un entorno llamado *deepLearning* (el nombre es a gusto de cada uno), ejecutamos el comando:

```
conda create -n deepLearning pip python=3.6
```

2. Activaremos el entorno creado con el comando:

```
activate deepLearning
 (deepLearning)C:>  # El prompt debería cambiar
```

3.2. Instalación de las librerías Deep Learning

Para instalar TensorFlow, en el *prompt Anaconda* ejecutaremos los siguientes comandos:

1. Para instalar la versión GPU de TensorFlow:

```
pip install tensorflow-gpu
```

2. Para instalar la versión CPU de TensorFlow:

```
pip install tensorflow
```

Para instalar Keras, en el *prompt Anaconda* ejecutaremos el siguiente comando:

```
pip install keras
```

3.3. Otras librerías necesarias

Además de las librerías TensorFlow y Keras, el sistema necesita otras librerías para su funcionamiento. Para instalar cualquiera de estas librerías hemos de proceder igual que con las anteriores. En el *prompt Anaconda* ejecutaremos el siguiente comando:

```
pip install <nombre_de_la_librería>
```

Las librerías necesarias son:

- OpenCV
- Pandas
- Matplotlib

- Numpy
- PIL

Anexo II. Uso de un entorno remoto para Deep Learning.

En este anexo se presenta la herramienta Google Colab, que permite la ejecución en la nube de *Jupyter Notebooks* con Python, así como las instrucciones a seguir para ejecutar con ella un sistema *Deep Learning*.

1. Google Colab

Google Colab es un servicio remoto gratuito basado en *Jupyter Notebooks*, que incluye aceleración de cálculos por GPU de forma gratuita, incluido en el servicio Google Drive. Colab permite además el desarrollo de aplicaciones Deep Learning utilizando algunas de las bibliotecas más populares como PyTorch, TensorFlow, Keras y OpenCV.

Actualmente el servicio soporta los núcleos (*kernels* en inglés) de Python 2.7 y 3.6, pero aún no soporta otros núcleos de Jupyter como R o Scala. Otras restricciones del servicio son el tiempo limitado de las sesiones y el tamaño de memoria permitida en GPU, pero incluso en esta versión limitada, se han podido entrenar todas las redes de este proyecto sin problema.

1.1. Creando un nuevo Notebook

Para crear un nuevo notebook de Colab, estando dentro de nuestro Google Drive, hemos de pulsar “Nuevo”, en el menú desplegable pulsamos “Más” y pulsamos en “Colaboratory”. Una vez creado el notebook podemos renombrarlo, copiarlo, modificarlo, compartirlo o descargarlo como cualquier otro archivo de Drive. También podemos subir a Drive notebooks que tengamos en nuestro disco duro utilizando la opción “Nuevo”, “Subir archivo”, o simplemente arrastrándolos a la carpeta de Drive en el navegador.

Una vez dentro de Colab, pulsando en “File”, “Open notebook” se nos abre el menú de la aplicación que nos permite abrir notebooks desde distintos sitios:

- Notebooks con ejemplos de la propia Google Colab.
- Notebooks recientes.
- Notebooks que se encuentren en nuestro Drive.
- Notebooks que se encuentren en GitHub.
- Subir Notebooks desde nuestro disco duro.

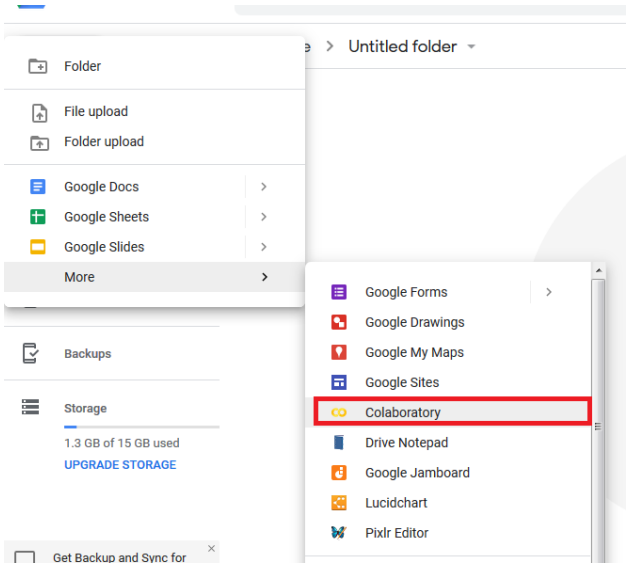


Ilustración 87 Creación de un Notebook en Google Drive (Fuente: Google Drive).

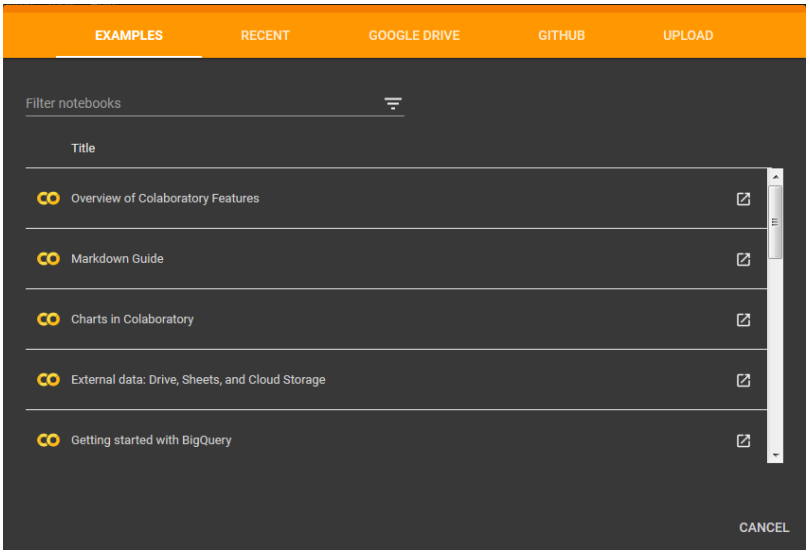


Ilustración 88 Menú de apertura de Notebooks en Google Colab (Fuente: Google Drive).

1.2. Activando la aceleración por GPU

Para activar la aceleración de cálculos por GPU para aquellas bibliotecas que lo permitan (como TensorFlow), simplemente hemos de pulsar en "*Runtime*" para abrir el menú desplegable y pulsar en "*Change runtime type*". En el menú que nos aparece podemos seleccionar si queremos un entorno de Python 2 o Python 3 y el tipo de aceleración que queremos: ninguna, por GPU o por TPU.

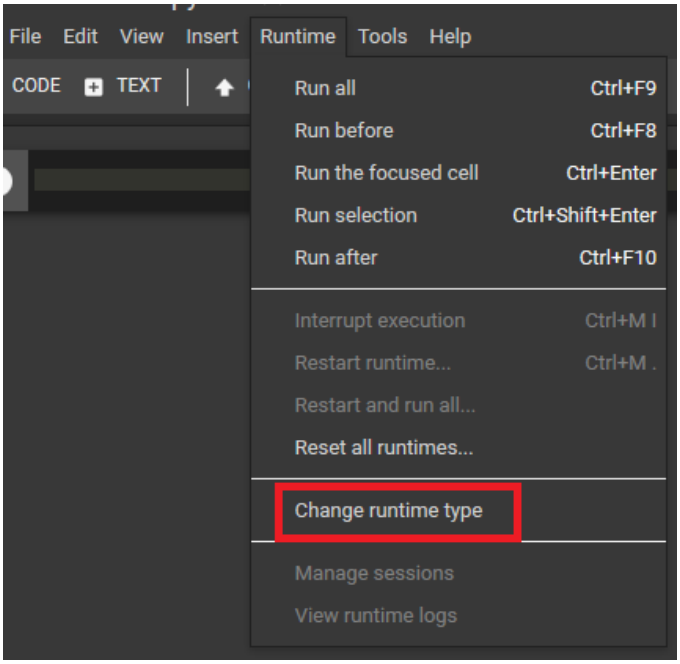


Ilustración 89 Apertura el menú de entorno de ejecución en Google Colab (Fuente: Google Drive).

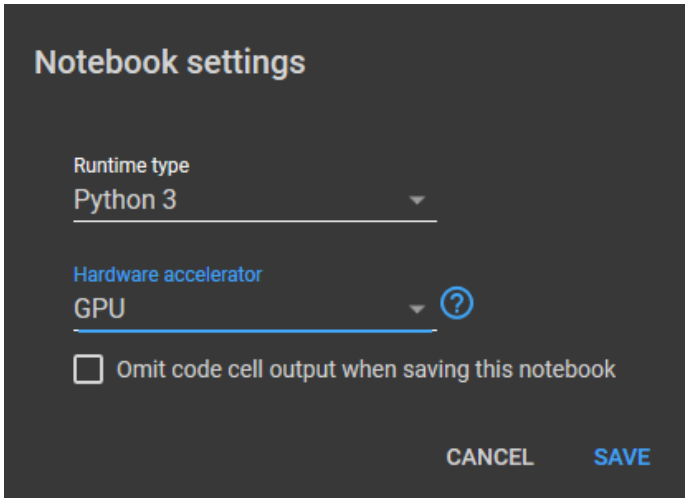


Ilustración 90 Selección del entorno de ejecución en Colab (Fuente: Google Drive).

1.3. Instalando las bibliotecas necesarias

Colab tiene instaladas por defecto algunas bibliotecas de Python, pero no todas. Para instalar aquellas que necesitemos para nuestro proyecto hemos de ejecutar los

mismos comandos que utilizaríamos en nuestro entorno local (como `pip install`), precedidos de un símbolo de exclamación (!). Por ejemplo, si queremos instalar Keras:

```
!pip install keras
```

1.4. Importando los datos para la red

Para importar la base de datos necesaria para entrenar y validar nuestra red neuronal tenemos varias opciones:

- Si ya tenemos la base de datos guardada en nuestro Google Drive podemos montar nuestro Google Drive como un disco duro virtual al que podemos acceder mediante la máquina virtual del entorno de ejecución. Para montar este disco hemos de ejecutar la siguiente celda en Colab:

  ```
  from google.colab import drive
  drive.mount('/content/gdrive')
  ```

  A continuación hemos de pulsar el enlace que nos aparece, permitir el acceso a nuestro Google Drive, copiar el código que nos aparece y pegarlo en la salida de la celda anterior, y pulsar “Intro”. Si todo funciona correctamente veremos algo similar a la siguiente imagen.

Ilustración 91 Montando nuestro Drive como disco virtual en Colab (Fuente: Google Drive).

  Ahora podemos acceder a los contenidos de nuestro Drive como si fuese un disco duro local.

  ```
  !ls "/content/gdrive/My Drive/"
  ```

- Si la base de datos está disponible para descargar de forma pública (en la plataforma *Amazon Web Services*, por ejemplo) podemos descargarlo a nuestro entorno virtual.

Para ello podemos utilizar los comandos `wget` y `unzip`.

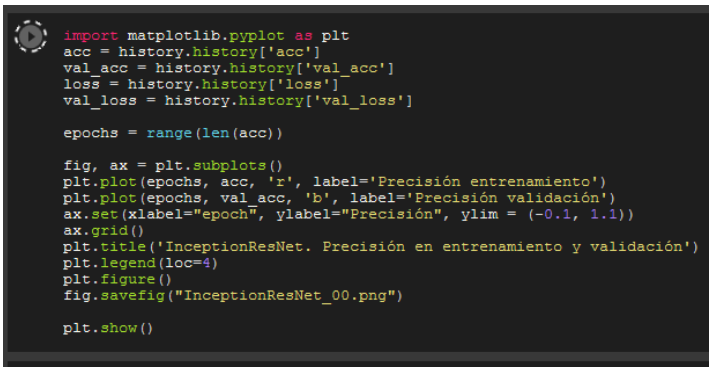
```
!wget -cq https://s3.amazonaws.com/content.udacity-
data.com/courses/nd188/flower_data.zip
!unzip -qq flower_data.zip
```

- Si disponemos de la base de datos en un repositorio remoto (como puede ser GitHub) podemos clonar dicho repositorio a nuestro entorno virtual.

```
!git clone --recursive
https://github.com/carlosteba/TFG_DeepLearning_Comics
```

1.5. Ejecutando nuestro código

Los notebooks de Colab funcionan igual que los notebooks que abrimos con la herramienta Jupyter Notebooks. Podemos introducir celdas de código Python o celdas de texto en lenguaje *Markdown* y ejecutar estas celdas pulsando en el icono “Play” ➤ o la combinación Ctrl + Enter.



```
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

fig, ax = plt.subplots()
plt.plot(epochs, acc, 'r', label='Precisión entrenamiento')
plt.plot(epochs, val_acc, 'b', label='Precisión validación')
ax.set(xlabel="epoch", ylabel="Precisión", ylim = (-0.1, 1.1))
ax.grid()
plt.title('InceptionResNet. Precisión en entrenamiento y validación')
plt.legend(loc=4)
plt.figure()
fig.savefig("InceptionResNet_00.png")

plt.show()
```

Ilustración 92 Celda en ejecución en Colab (Fuente: Google Drive).

Anexo III. Ficheros de código y base de datos del proyecto

En este anexo se explican brevemente los contenidos de las distintas carpetas y ficheros que acompañan a este proyecto, para aquellas personas que deseen utilizar la base de datos creada o los distintos ficheros de código para la creación de la base de datos o para el entrenamiento de las distintas redes neuronales.

1. Ficheros de código

En la carpeta `código` se encuentran todos los ficheros necesarios para ejecutar las distintas etapas de creación de la base de datos y las distintas redes neuronales utilizadas en el proyecto.

Ilustración 93 Jerarquía de carpetas del código del proyecto

A continuación, se detallan los contenidos de cada una de las carpetas de la jerarquía anterior:

- **`Recopilacion_imagenes`**: aquí se encuentra el script de Python para descargar imágenes utilizando la API de Bing Search, descrito en el apartado 4.4.1.
- **`Preprocesamiento`**: en esta carpeta se encuentran todos los scripts de Python para realizar el preprocesamiento de las imágenes (recortado, redimensionado, etiquetado, etc.). Además se ha incluido un *Notebook* de *Jupyter* (`preprocesamiento.ipynb`) en el que se puede realizar todo el proceso de descarga y procesado de las imágenes.
- **`Pruebas_preliminares`**: aquí están los 11 *Jupyter Notebooks* con los que se han realizado las pruebas preliminares descritas en el apartado 6.1. Tres *notebooks* para las redes sin modificaciones, tres para el aumento de datos,

dos para la transferencia de conocimiento y otros tres para las pruebas con las imágenes de tamaño 299 x 299.

- **`Busqueda_cuadricula`**: en esta carpeta se encuentran los 4 Jupyter Notebooks utilizados para los 4 experimentos descritos en el apartado 4.2.
- **`Modelo_final`**: en esta carpeta se encuentra el Jupyter Notebook con el modelo utilizado para las predicciones finales sobre el conjunto de test, descrito en el apartado 4.3.

Además, se incluyen los ficheros *etiquetas.csv*, que contiene las etiquetas de cada una de las instancias de la base de datos y *util.py*, que contiene varias funciones para leer las imágenes y las etiquetas y organizarlas en las estructuras de datos necesarias para las redes neuronales.

2. Ficheros de la base de datos

En la carpeta `dataset` se encuentra la base de datos de imágenes utilizada, incluyéndose, no solamente la versión final de la base de datos, sino también las imágenes originales descargadas, así como las imágenes en cada paso del proceso de preprocesamiento.

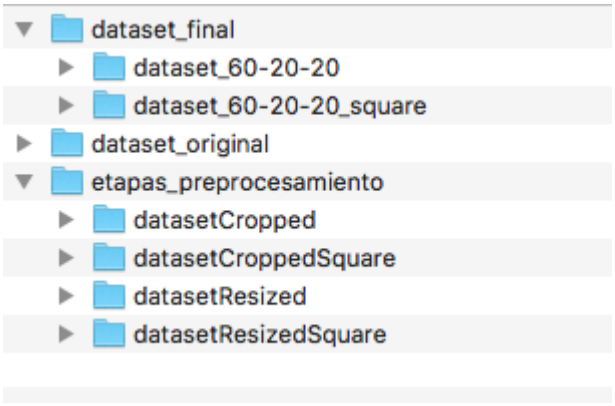


Ilustración 94 Jerarquía de carpetas de la base de datos.

A continuación, se detallan los contenidos de cada una de las carpetas de la jerarquía anterior:

- **`dataset_60-20-20`**: aquí se la versión final de las imágenes de tamaño 135 x 225 píxeles, divididas en tres carpetas, para **entrenamiento** (60% de las imágenes), **validación** (20%) y **test** (20%).
- **`dataset_60-20-20_square`**: igual que la anterior, salvo que aquí se encuentran las imágenes de tamaño 299 x 299 píxeles.

- **`dataset_original`**: esta carpeta contiene las imágenes originales descargadas con la API de Bing Search, separadas en carpetas, una para cada uno de los cómics descritos en la Tabla 2.
- **`datasetCropped`**: en esta carpeta están las imágenes recortadas según la razón 3:5, separadas por cómics, igual que en la carpeta anterior.
- **`datasetCroppedSquare`**: igual a la anterior, pero con las imágenes recortadas a la razón 1:1.
- **`datasetResized`**: aquí están las imágenes redimensionadas a tamaño 135 x 225 píxeles antes de ser divididas en los subconjuntos de entrenamiento, validación y test.
- **`datasetResizedSquare`**: igual a la anterior, pero con las imágenes de tamaño 299 x 299 píxeles.