







# Resumen

Actualmente, la mayoría de modelos de clasificación de imágenes digitales está basado en el uso de redes neuronales convolucionales, debido a su demostrada mejora respecto a los modelos de machine learning tradicional basados en extracción de características.

Sin embargo, existen problemas para los que los resultados obtenidos en base a un entrenamiento de un modelo base no ofrecen los mejores resultados. Para poder mejorarlos, existen técnicas que modifican bien el conjunto de entrenamiento, el propio entrenamiento de los modelos o incluso la forma de evaluación.

El objetivo de este TFG consiste en realizar una búsqueda y análisis de dichas técnicas y estudiar las posibles mejoras de rendimiento en base a su utilización. Se estudiarán varios conjuntos de datos para comprobar la utilidad de cada una de las técnicas.

# 

# Palabras clave

* visión artificial
* aprendizaje automático supervisado
* redes neuronales convolucionales
* estado del arte
* label smoothing
* data augmentation
* test time augmentation
* clasificación de imágenes

# Índice

[Resumen](#_c65todqwuvp1)

[Palabras clave](#_64yzcoa056gs)

[Índice](#_q326xdv33geb)

[Capítulo 1 - Introducción y presentación de objetivos](#_58qmamj7t7dt)

[Capítulo 2 - Qué son las Redes Neuronales Convolucionales](#_jkejpgigspl9)

[Capítulo 2.1 - Capa convolucionales](#_il59xxmp0y1b)

[Capítulo 2.2 - Capas de condensación](#_hgs7gnfawmff)

[Capítulo 2.3 - Capas dropout](#_a3le2xbn8l8l)

[Capítulo 2.4 - Capa de normalización](#_unhq29xlfcdu)

[Capítulo 2.5 - Fase de clasificación](#_8qbo23dhw4zr)

[Capítulo 3 - Técnicas de mejora de entrenamiento y evaluación](#_pr1mjdv9ggef)

[Capítulo 3.1 - Label smoothing](#_w6xsclpmn28g)

[Capítulo 3.2 - Data Augmentation](#_1bl6hs2zbe00)

[Capítulo 3.3 - Test time augmentation](#_yh5gwh7c67x1)

[Capítulo 4 - Experimentos realizados](#_kvrnz12qrxiv)

[Capítulo 4.1 - Dataset elegido](#_4iwr538nr7xd)

[Capítulo 4.2 - Entorno de desarrollo](#_lrb67g55gprh)

[Capítulo 4.3 - Obtención de datos](#_mxrqomh5gmyx)

[Capítulo 4.4 - Transfer learning](#_2gxzjxytchk7)

[Capítulo 4.5 - Fine tuning](#_k368v24e3tla)

[Capítulo 4.6 - CNN personalizada](#_eeagdnaghput)

[Capítulo 4.7 - Data augmentation](#_dhzlz2jqfz36)

[Capítulo 4.8 - Label smoothing](#_j9ml1kv5777h)

[Capítulo 5 - Resumen de resultados obtenidos](#_z8d8ez38m1sk)

[Capítulo 5.1 - Transfer learning con una ResNet 50](#_4eqyhalgsq7a)

[Capítulo 5.2 - Fine tuning de la ResNet 50 anterior](#_fxipp12p6u4y)

[Capítulo 5.3 - Usando una CNN personalizada sin modelo pre-entrenado](#_dpdqfk2x2t4z)

[Capítulo 5.4 - Data augmentation](#_aoa9k8yz4343)

[Capítulo 5.5 - Label smoothing](#_nx6clfg4lz0n)

[Capítulo 5.6 - Test Time Augmentation](#_f7tl2q7sjnzp)

[Capítulo 6 - Conclusiones y líneas futuras](#_54t5wotlpx4d)

[Anexos](#_l2vq04jribs8)

# Capítulo 1 - Introducción y presentación de objetivos

La historia de las redes neuronales convolucionales (CNN) se remonta a los años 80, cuando el científico Yann LeCun introdujo una arquitectura conocida como LeNet-5, la cual sentó las bases para el desarrollo posterior de las CNN. En ese momento, la computación y los recursos disponibles limitaron su aplicación práctica.

Sin embargo, a medida que avanzaban las décadas, los avances tecnológicos y el aumento en el poder de procesamiento de las computadoras permitieron el resurgimiento y la popularización de las CNN. A principios de los años 90, LeCun y su equipo utilizaron las redes neuronales convolucionales para realizar reconocimiento de dígitos manuscritos en cheques bancarios, lo que resultó en un gran avance en el campo del procesamiento de imágenes.

En la década del 2000, las CNN experimentaron un crecimiento significativo gracias a la aparición de grandes conjuntos de datos anotados, como el conjunto de datos ImageNet, que contiene millones de imágenes etiquetadas en diferentes categorías. Este conjunto de datos fue utilizado para entrenar y evaluar modelos de CNN en un desafío llamado ImageNet Large Scale Visual Recognition Challenge (ILSVRC), el cual impulsó el desarrollo y la mejora de las redes neuronales convolucionales.

Uno de los hitos más destacados en la historia de las CNN se produjo en 2012, cuando el equipo de investigación de Geoffrey Hinton, utilizando una arquitectura llamada AlexNet, ganó el primer lugar en el ILSVRC, superando por un margen considerable a los enfoques tradicionales. AlexNet constaba de múltiples capas convolucionales y de submuestreo, y fue capaz de aprender características jerárquicas complejas en las imágenes, revolucionando el campo del reconocimiento de imágenes.

A partir de ese momento, las CNN se volvieron omnipresentes en diversas aplicaciones de visión por computador, como el reconocimiento facial, la detección de objetos, la segmentación de imágenes y el procesamiento de videos. Surgieron nuevas arquitecturas como VGG, GoogLeNet, ResNet y DenseNet, que incorporaron capas más profundas y técnicas como la conexión residual para mejorar el rendimiento y la capacidad de generalización de las CNN.

Además, las CNN han demostrado su utilidad en otras áreas, como el procesamiento de texto y el análisis de datos secuenciales. Por ejemplo, se han aplicado a tareas como la traducción automática, el reconocimiento de voz y el análisis de sentimientos en textos.

En resumen, la historia de las redes neuronales convolucionales ha sido una evolución constante desde su concepción inicial en los años 80 hasta convertirse en una de las herramientas más poderosas y populares en el campo del aprendizaje automático y la visión por computadora.

Gracias a su capacidad para extraer características relevantes de las imágenes y aprender patrones complejos, las CNN han impulsado avances significativos en el reconocimiento y la comprensión de imágenes, y continúan siendo objeto de investigación y desarrollo en la actualidad. Usándose para detectar diversos tipos de tumores automáticamente, reconocimiento de vídeo, tareas de procesamiento del lenguaje natural o enseñar a conducir a los coches autónomos por ejemplo.

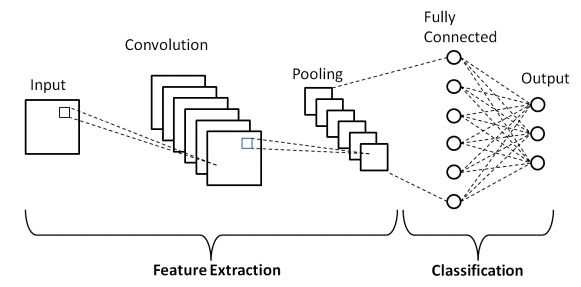
Sin embargo para entrenar CNNs se necesita invertir mucho en obtención de imágenes y recursos hardware. Una estrategia de mejora es aumentar el tamaño de los modelos, lo cual requiere de más tiempo y potencia de procesamiento, corriendo el riesgo de sobre ajustarse a los ejemplos con los que entrenamos. Esto se puede paliar con capas de regularización como las L1 o L2 o capas de dropout.

Otro enfoque es mejorar los datos utilizados para entrenar pero necesita de dinero, tiempo y una estrategia clara bien sea de data cleaning o de recogida de nuevos datos.

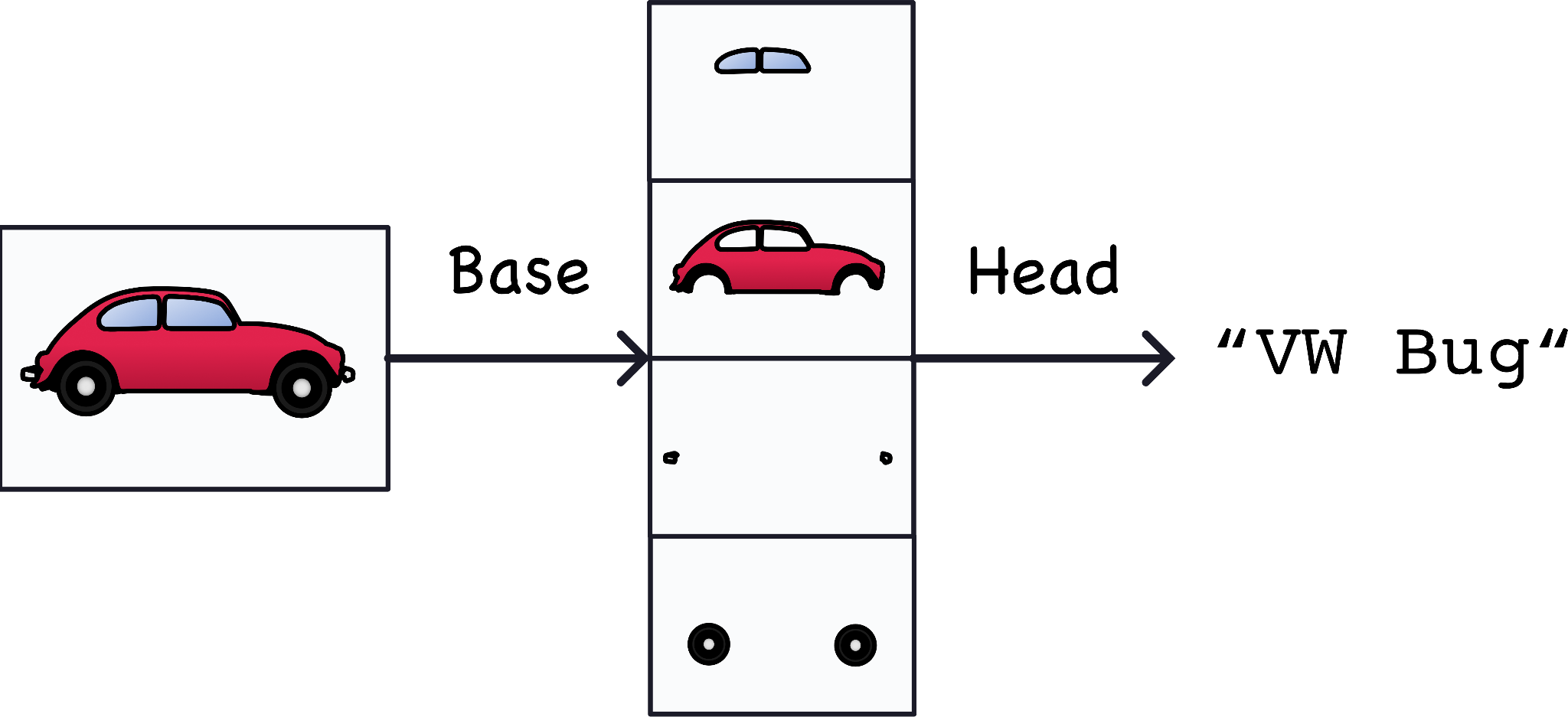
Por suerte, hay técnicas que nos pueden ayudar a obtener mejores modelos sin necesidad de invertir dinero y tiempo en aumentar los datos o los recursos. El objetivo de este TFG es presentar algunas de las técnicas más utilizadas con este fin y ponerlas a prueba con un problema real.

# Capítulo 2 - Qué son las Redes Neuronales Convolucionales

Las Convolutional Neural Networks son redes multi capa que toman su inspiración del córtex visual de los animales. Esta arquitectura es útil en varias aplicaciones, principalmente procesamiento de imágenes. Están compuestas por dos partes: la base donde se extraen características de las imágenes y la cabeza donde se clasifican.



Las características van desde líneas, colores, vértices, texturas, formas a patrones o combinaciones de ellas.

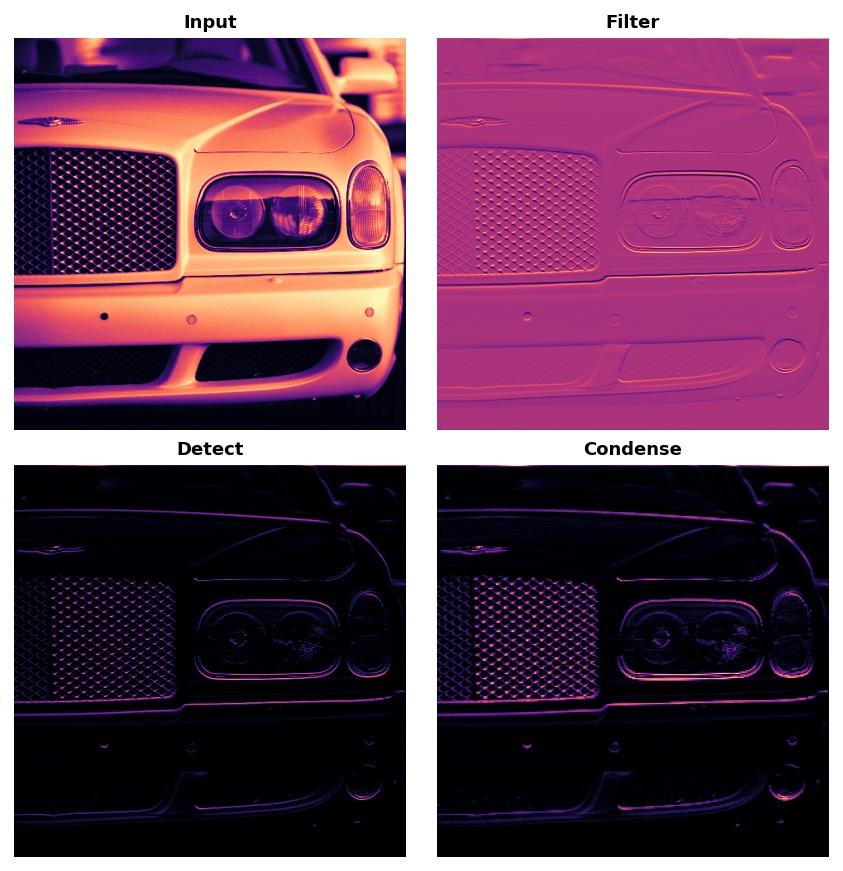


Al entrenar CNNs queremos mejorar en uno de dos aspectos: que nuestra base extraiga mejores características o relacionar más unívocamente esas características a una clase.

La arquitectura consta de varias capas. Estas son las principales: convolucionales, de condensación como MaxPooling, de regularización como las dropout y de normalización.

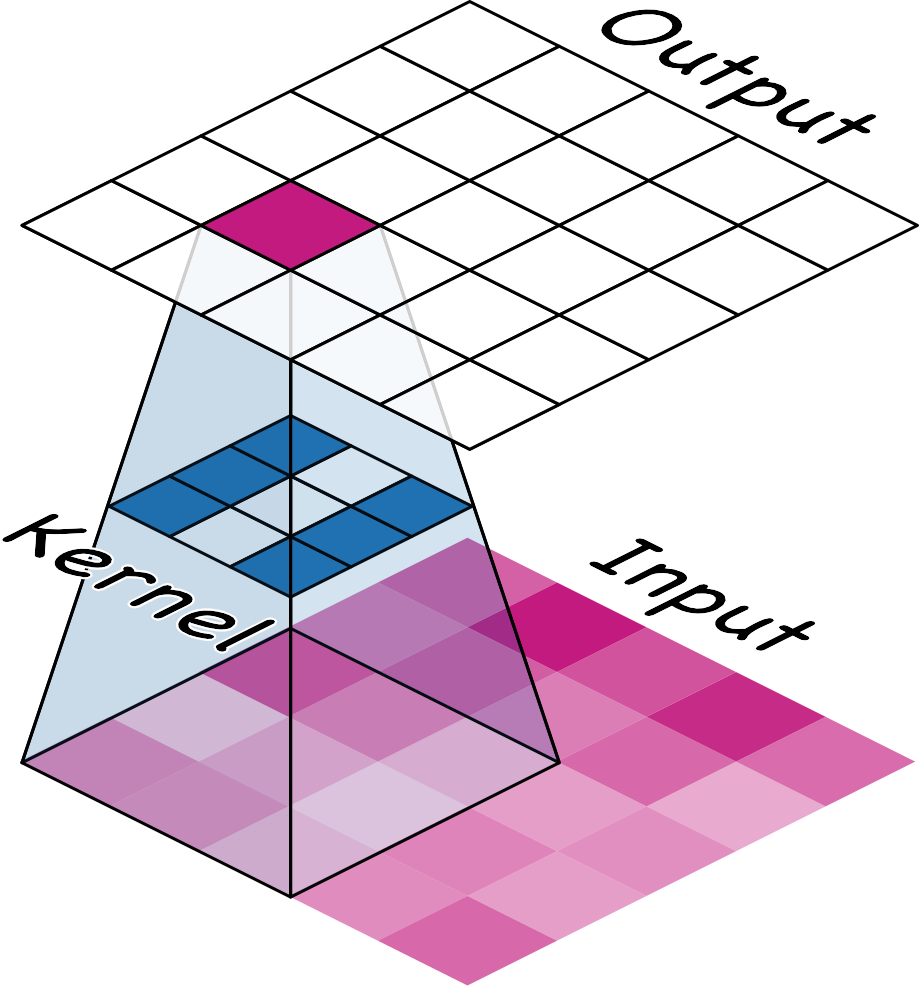
## Capítulo 2.1 - Capa convolucionales

Entremos un poco en detalle de cómo centran el foco las bases en unas características u otras. Con tres operaciones básicas: filtrar, detectar y condensar conseguimos el objetivo.



En el caso de esta imagen se aíslan las líneas horizontales, esta es la salida de una sola neurona y de una sola capa. Por lo que en una misma capa se avistan más propiedades de la imagen y a su vez esas características son sintetizadas por la siguiente capa.

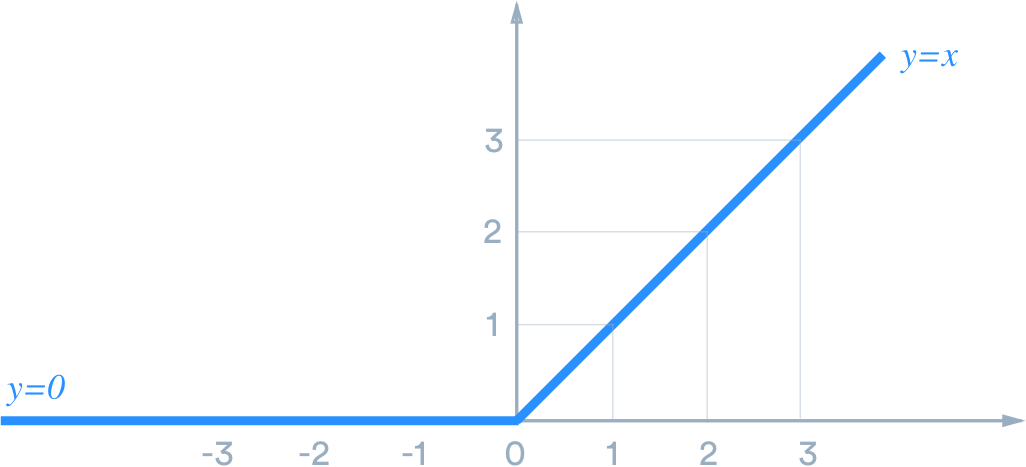
De la fase de filtrado se encargan las capas convolucionales que dan nombre a la red. Están compuestas por kernels: matrices normalmente cuadradas también conocidas como filtros, los cuales recorren las imágenes que entran a cada neurona de arriba a abajo y de izquierda a derecha. Enfatizan o restan importancia a diversos pixeles, además de juntar información de pixeles con sus vecinos en la característica resultante.



En la fase de entrenamiento lo que se busca aprender en estas capas son los pesos de los kernels, es decir los valores que conforman esas matrices.

Para personalizar estas capas podemos jugar con el tamaño de los filtros siendo normalmente números impares tanto de filas como de columnas, aunque no es un requisito obligatorio; con cuantas neuronas conforman la capa o lo que es lo mismo cuantas características extraemos por capa; el padding usado ya que si no usamos las imágenes se irán haciendo más pequeñas a lo largo de la red; o cuantos pixeles movemos la ventana deslizante que realiza la convolución en cada paso.

Para la fase de detección se usan las funciones de activación de las neuronas. Para clasificación el estándar es usar la función RELU, la cual asigna a todos los píxeles negativos un cero.



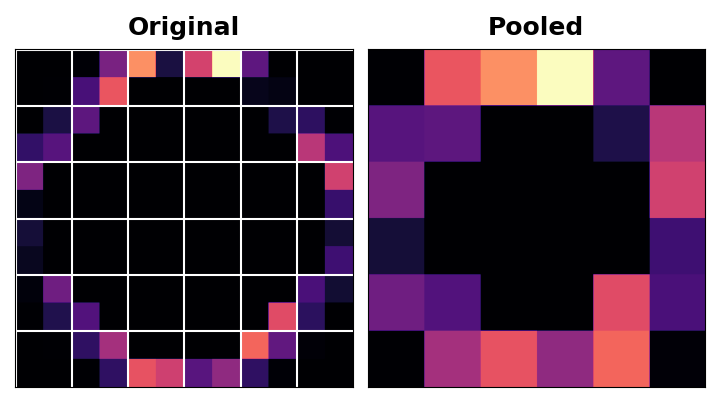
Esto se hace para dar la misma importancia a todos los pixeles no importantes, en este caso los negativos.

La RELU es no lineal, como la gran mayoría de las funciones de activación, lo que permite que las características extraídas con los filtros se combinen de manera más interesante a lo largo de la red.

## 

## Capítulo 2.2 - Capas de condensación

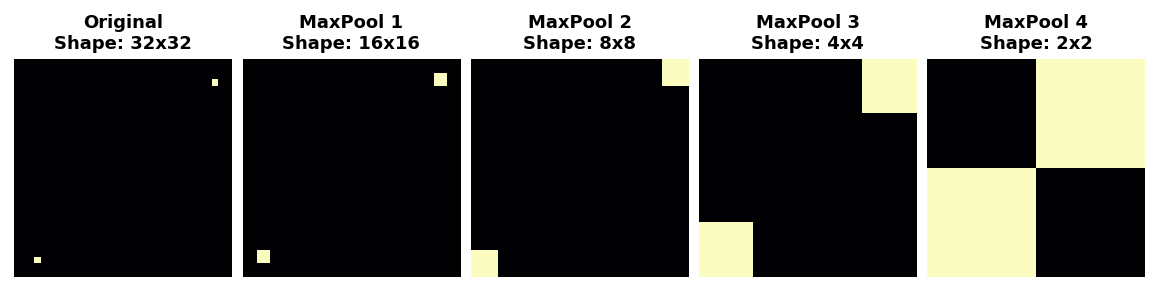
El siguiente paso es el pooling que reduce la dimensionalidad de las imágenes extraídas manteniendo la información más importante. MaxPooling por ejemplo se queda con el máximo de los vecinos, es una idea parecida a la capa convolucional pero los parámetros de estos kernels no son entrenables ni lineales.



El pooling al decrementar mucho el tamaño de las imágenes hace que no trabajemos con tanta información redundante e inutil como pixeles en negro o lo que es lo mismo a cero. De esta manera las operaciones en capas posteriores son mucho más rápidas. También se suele utilizar la media de los vecinos entre otros.

Al aplicar MaxPooling después de la función de activación lo que hacemos es incrementar la proporción de píxeles positivos frente a píxeles a cero. De esta manera intensificamos las características extraídas en los pasos anteriores.

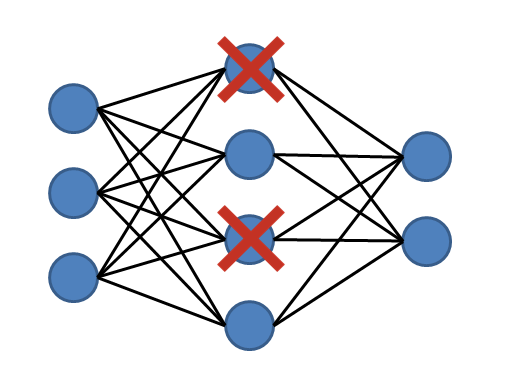
Pero no todo es bueno porque los pixeles a cero nos dan información posicional de los objetos en la imágen, por lo que al hacer MaxPooling perdemos parte de esa información en pro de mayor velocidad y espacio en RAM.



Esta última desventaja del MaxPooling puede ser de hecho una ventaja, ya que dependiendo del conjunto de datos dos fotos de la misma clase en las que los objetos están trasladados por la foto sin rotación querríamos que se clasifiquen igual. MaxPooling hace esta abstracción de forma natural generalizando bien en este tipo de casos. No tenemos que entrenar a la red para ignorar este tipo de diferencias.

## Capítulo 2.3 - Capas dropout

Estas capas son las principales que podemos añadir, pero no las únicas. Las capas de dropout son bastante usadas en las CNN para evitar el sobreaprendizaje. Las redes neuronales tienden a buscar formas muy enrevesadas y específicas de relacionar los datos con los que entrenan. Formando teorías frágiles que no generalizan bien, para evitar este patrón la idea tras las capas dropout es romper aleatoriamente algunos enlaces entre neuronas.



Por así decirlo con un símil es como si la teoría que forma nuestra CNN fuese una torre de palillos y nosotros le quitamos palillos para ver si aguanta y por lo tanto es una hipótesis fuerte o por si lo contrario no es lo suficientemente estable, se cae y más adelante en el entrenamiento tenemos que desechar esa conjetura.

En cada epoch o vuelta al dataset rompemos unos enlaces distintos por lo que al final del entrenamiento nos quedamos con el conocimiento más fuerte y extrapolable al “mundo real”.

## Capítulo 2.4 - Capa de normalización

Otro tipo de capa muy utilizada es la normalización por lotes o batch normalization. Se usa para manejar lotes de datos con la misma escala dentro de la red, ya que las imágenes con píxeles más altos suelen tener más peso y de esta manera deshacemos ese sesgo.

## 

## Capítulo 2.5 - Fase de clasificación

La cabeza de la CNN tiene que ser capaz de convertir el mapa de características extraídas en la fase anterior a bien sea un valor en caso de regresión o tantas probabilidades como clases haya en los problemas de clasificación. Para ello la solución más habitual que se usa es pasar todas las características a vectores y aplicar capas densas.

También es usual sustituir la vectorización de las imágenes por un global average pooling que supone tomar como representante de cada imagen la media de sus píxeles, esta última versión aunque menos habitual reduce mucho los parámetros usados en la fase final de la red.

# Capítulo 3 - Técnicas de mejora de entrenamiento y evaluación

## Capítulo 3.1 - Label smoothing

One hot encoding es una codificación en la que se usa tantos enteros como clases haya, asignando cero a todos menos a aquel en la posición equivalente a la clase a la que pertenece el ejemplo que lo ponemos a uno.

Aunque se evita usar para ahorrar memoria, cuando la usamos entrenamos un modelo para el cual la predicción esperada va a estar también en one hot encoding. Para la clase que predice va a devolver un 100% de probabilidad de que sea esa clase y para el resto un 0%, transmitiendo una falsa seguridad. No es muy realista esta forma de operar porque seguramente habrá casos en los que no esté tan seguro de a qué clase corresponde el ejemplo, sobre todo en conjuntos de datos donde el etiquetado de los ejemplos sea bastante mejorable.

Por lo que le queremos obligar a que devuelva respuestas más conservadoras y con menos certeza. Para ello partimos de one hot encoding y cambiamos los ceros por siendo N el número de clases y el hiper-parámetro a elegir, cuanto mayor más label smoothing aplicamos por lo tanto el modelo devolverá respuestas con más dudas.

Por ejemplo significa que estamos 10% seguros de nuestras etiquetas. El uno inicial lo sustituimos por . En un caso con 10 clases pasariamos de esto:

**[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]**

a esto:

**[0.01, 0.01, 0.01, 0.91, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01]**

## Capítulo 3.2 - Data Augmentation

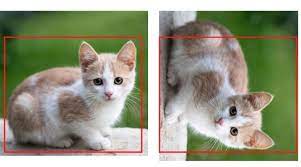
La práctica hace la perfección. En muchos casos es imposible alcanzar la perfección pero queremos quedarnos tan cerca como podamos. Es por eso que cuanto más ejemplos vea nuestro modelo mejor. Pero, ¿Qué podemos hacer si ya hemos usado todo el conjunto de datos? Data augmentation palia este problema. Se trata de una técnica, como su nombre dice, de añadir ejemplos nuevos.

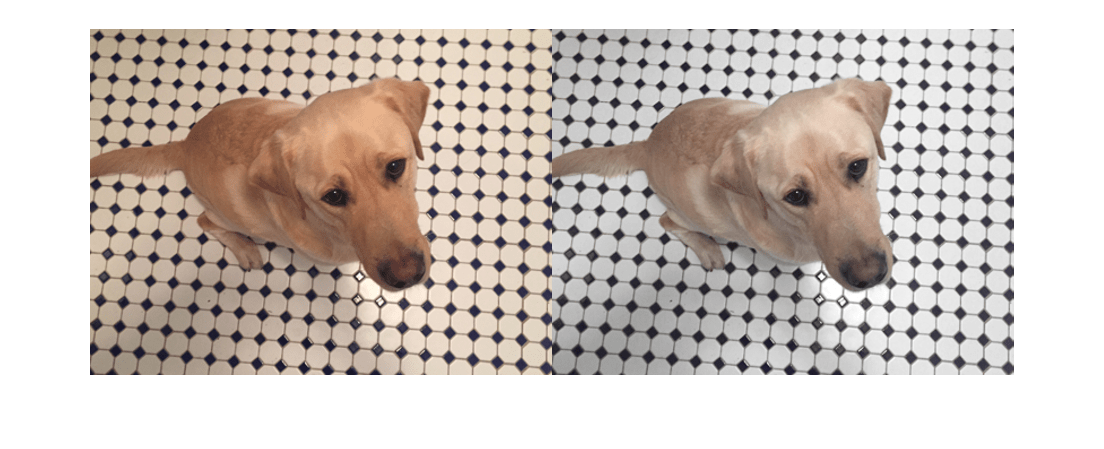
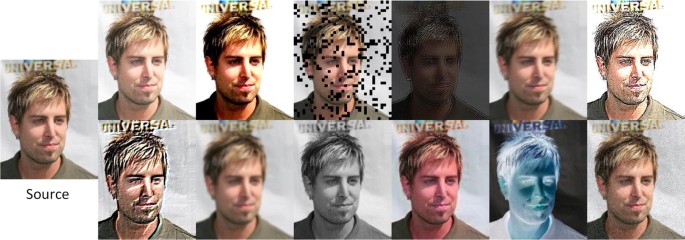
Más o menos, porque en realidad cogemos los ejemplos que ya hemos utilizado y les aplicamos pequeños cambios para que la red piense que son nuevas imágenes sin tener que invertir más tiempo y dinero en aumentar el dataset.

De esta forma conseguimos que el modelo generalice mejor y por lo tanto somos capaces de entrenar más epochs sin caer en sobreaprendizaje, lo que se traduce últimamente en un mejor modelo final.

Pero no podemos aplicar cualquier tipo de cambios, tienen que tener sentido en el problema en específico, ya que sino conseguiremos el efecto contrario al buscado.

Algunas de las transformaciones más vistas son: rotaciones, inversión horizontal, inversión vertical, cambio en el contraste, cambio en brillo, traslación horizontal, traslación vertical, recorte de bordes, entre otras.



## Capítulo 3.3 - Test time augmentation

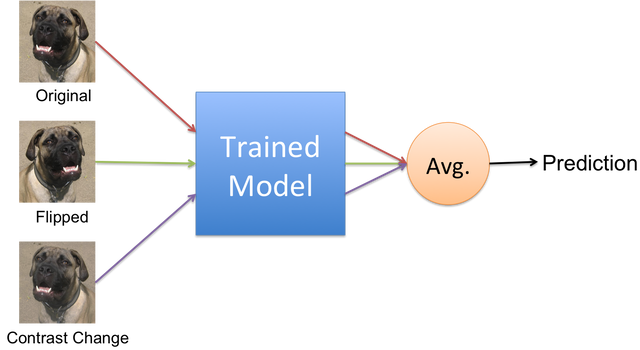
Hilando con el concepto anterior que nos permite entrenar un mejor modelo podemos equivocarnos en imágenes un tanto dudosas. Los humanos tendemos a equivocarnos también y una forma de evitar esto es formando comités de personas que den su punto de vista. ¿Qué es mejor que un buen modelo? Varios modelos buenos.



Para imitar este razonamiento lo que podemos hacer es para cada imagen que tenemos que clasificar aplicarle las mismas transformaciones que hemos usado al entrenar pero en este caso en la fase de predicción una vez el modelo ya está en producción.

Para ello cogemos la imagen que nos llega y predecimos además de la original todas sus transformaciones independientemente, luego agregamos esos resultados y de ahí obtenemos una clase más fiable de la que hubiésemos predicho simplemente asignando una clase a la imagen original.

Tenemos varias formas de agregar las predicciones, la más habitual es hacer la media de las probabilidades obtenidas por cada imagen.

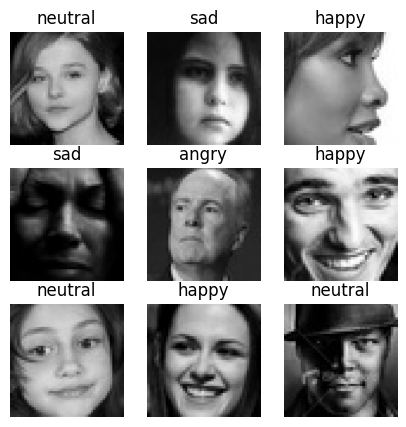


# Capítulo 4 - Experimentos realizados

## Capítulo 4.1 - Dataset elegido

Para aprovechar todo este conocimiento tenemos que ponerlo en práctica. No hay mejor prueba que un caso real. Para ello he escogido el dataset FER-2013, Face Emotion Recognition, el cual nos brinda datos con los que entrenar y testar un modelo de reconocimiento de emociones faciales.

Todas las imágenes son de 48x48, en escala de grises, vienen todas las caras centradas parecido en la imagen y hay siete clases: angry, disgust, fear, happy, sad, surprise y neutral. Hay 28708 ejemplos de entrenamiento-validación y 7178 de test.



Como se puede observar en la muestra las emociones son un poco ambiguas ya que una misma foto podría ser clasificada en dos emociones distintas y ser correcto.

Por eso he buscado unas etiquetas que concuerdan más con este problema, pero a su vez mantener las originales para ver que tal funcionan las técnicas con mejores y peores etiquetas.

Es ahora cuando entran en juego las etiquetas FER-2013+. Las cuales han sido asignadas por un grupo de 10 personas, cada una vota a una clase.



Cada imagen como podemos ver puede tener votos en más de una clase, lo que hace que las etiquetas se asemejen más a las necesidades de este problema, siendo una solución más realista.

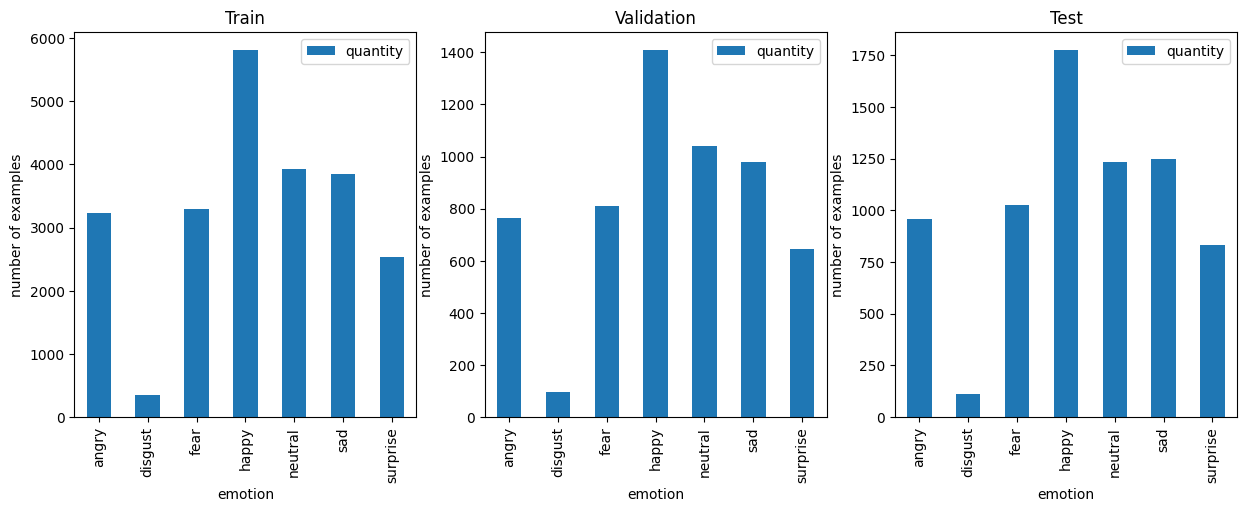
Además como en el dataset hay imágenes que no son caras se han añadido 3 clases más.



Las nuevas clases son: contempt, unknown y NF (not a face). Estas nuevas clases permiten eliminar fácilmente los ejemplos que no son caras, procedimiento que con las etiquetas originales se complica.

Por lo que las pruebas realizadas con unas etiquetas y con otras también tienen la diferencia de que en el dataset de FER-2013+ he limpiado el conjunto tanto de entrenamiento como de test, así que será interesante ver también cómo se adaptan las técnicas a ese tipo de problemas.

Otro aspecto a tener en cuenta del dataset es la distribución o representación de las clases, es un problema desbalanceado.



## Capítulo 4.2 - Entorno de desarrollo

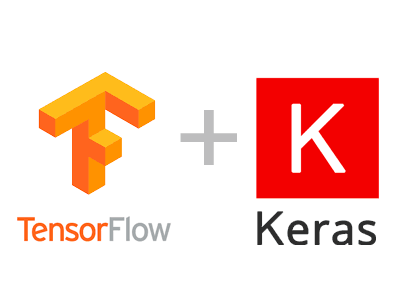
Para llevar a cabo las pruebas he usado python como lenguaje de programación porque estoy familiarizado con él ya que durante la carrera lo he utilizado en varias asignaturas, lo cual agiliza mucho el desarrollo de soluciones.

Además python es un lenguaje altamente utilizado para la visión artificial, por lo que cuenta con gran apoyo por parte de la comunidad en forma de librerías y documentación.



También he decidido usar Tensor Flow, más concretamente Keras para la confección de los modelos. Otras buenas opciones serían Fast.ai o Pytorch.

Me he decantado por Keras porque tenía un poco de experiencia a diferencia de con Pytorch. Fast.ai no permite tanta personalización como los otros dos.



Para agilizar los procesos de entrenamiento he activado el uso de GPU en mi ordenador personal. Tengo un Mac con chip M1 el cual ahora mismo solo permite usar un núcleo de GPU al mismo tiempo de los ocho que tiene porque las dependencias actuales no dejan activar los demás.

Alcanza una velocidad superior que entrenando por CPU pero aún así con modelos grandes es lento. Esto ha llevado a tomar decisiones más adelante en base a esta condición, ya que los procesos de entrenamiento se alargan bastante.

## **Capítulo** 4**.3 -** Obtención de datos

El procedimiento para FER-2013 y FER-2013+ ha sido un tanto distinto ya que la organización de los datos era diferente.

Empecemos con la carga en memoria de FER-2013. El dataset viene organizado en carpetas. Una carpeta grande contiene dos subcarpetas: train y test. Estas a su vez tienen siete carpetas, una por cada clase.

Para simplificar la obtención de imágenes y etiquetas he usado una pipeline que recorre las subcarpetas asignando a cada imagen la clase perteneciente a la carpeta en la que se encuentra. Posterior a esto los ejemplos los remuevo para que no estén organizados por clases y de la carpeta de train me guardo un 20% para la validación. De esta manera ya tengo tres conjuntos: train, validación y test. Los tres con ejemplos de todas las clases.

Luego aplico a los tres montones de datos una conversión del tipo de los pixeles a decimal, para poder más adelante normalizar las imágenes y que tengan media 0 y desviación estándar aproximadamente 1. Además organizo los datos en lotes de 128 y los optimizo para que keras los procese más rápido.

Ya que más adelante vamos a hacer uso de label smoothing me guardo las etiquetas codificadas en one hot encoding.

Como también vamos a utilizar modelos pre-entrenados con imágenes en 224x224 me guardo dos copias de los datos una con imágenes de tamaño 48x48, más ligeras y rápidas de procesar, para mis modelos entrenados desde cero y otra con las imágenes de 224x224.

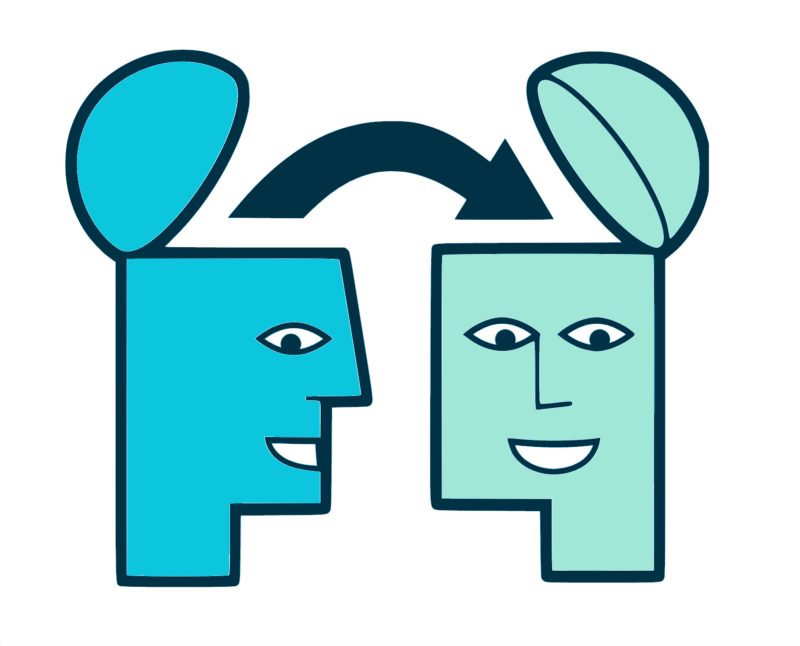
Para empezar a usar FER-2013+ tuve que fusionar dos csv y quedarme con lo que me interesaba. Esto se debe a que FER-2013+ es un csv que asigna nombres de imágenes a sus nuevas etiquetas. Cada fila es una imagen con 11 filas: el nombre de la imagen y 10 columnas con los votos que tiene esa clase en esa foto. Para organizarlo algo mejor fusione los pixeles serializados en una columna con las 10 columnas de las clases. Además separe el csv principal en tres: train, validación y test con una organización parecida a la obtenida en FER-2013.

Como la estructura actual en FER-2013+ me permite eliminar fácilmente aquellos ejemplos que son "unkown" o "not a face" lo hice. También normalice los pixeles de las imágenes, separe los datos en lotes de 128 y los optimicé para su procesamiento por modelos de Keras.

## **Capítulo** 4**.4 - Transfer learning**

El primer modelo que he decidido entrenar es en realidad uno que ya estaba pre-entrenado con imágenes externas al dataset con el que estamos trabajando. Como cuerpo de la red, es decir la parte que extrae características de las imágenes he escogido la ResNet de 50 capas versión 1. Se trata de una red residual con 48 capas convolucionales, una MaxPool y una AveragePool. Se llaman redes residuales porque apilan bloques residuales.

Para adaptar la ResNet 50 a nuestro caso he usado transfer learning, es una técnica con la que transmitir conocimiento que funciona para un problema a otro problema. En este caso la aplicación ha sido dotar de una cabeza nueva a la ResNet y entrenar tanto a la cabeza como las conexiones con la ResNet.



He optado por la cabeza más simple posible para ver desde donde partimos: una capa densa con tantas neuronas como clases hay, es decir 7.

## **Capítulo** 4**.**5 **-** Fine tuning

Otro concepto bastante potente y popular a la hora de usar bases pre entrenadas como ResNet es el fine tuning. Consiste en descongelar la fase final de la base.

En otras palabras en el capítulo anterior simplemente unimos la base con la cabeza y entrenamos tanto cabeza como las conexiones, al hacer fine tuning también entrenamos con nuestros datos las últimas capas de la base. Esto permite una mejor integración con la cabeza personalizada que usemos y por tanto que los conocimientos de la red sean más específicos del problema en concreto.

Esta técnica ofrece muchos mejores resultados que simplemente incrustando la cabeza como anteriormente. Pero a cambio necesita de muchos más recursos ya que una red con la ResNet de 50 capas tiene muchos parámetros a entrenar, incluso aunque solo tengamos que entrenar la parte final de la base la demanda aumenta mucho y por lo tanto el tiempo de entrenamiento también.

Me hubiese gustado usar la ResNet de 50 capas fine tuned durante todas las pruebas que he realizado, pero no ha sido viable por tiempo. Se hubiesen alargado demasiado las pruebas y el objetivo de este trabajo es comparar diferencias de modelos antes y después de usar las técnicas expuestas anteriormente.

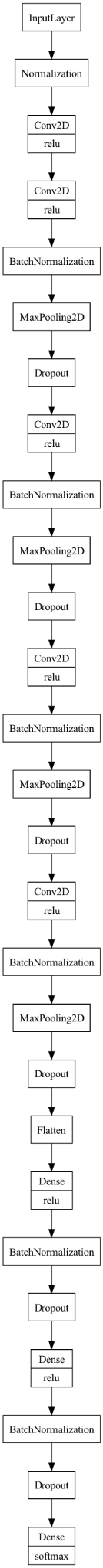
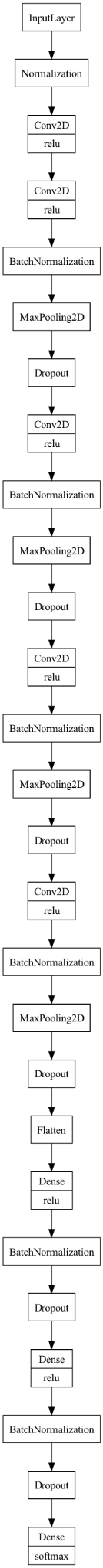
Si solo decidimos entrenar la cabeza del capítulo anterior que es la más simple que se puede hacer tenemos aproximadamente catorce mil parámetros a entrenar. Con fine tuning podemos descongelar capas hasta tal punto de tener veintitrés millones y medio de parámetros.

## Capítulo 4.6 - CNN personalizada

En el aprendizaje automático la velocidad y el número de entrenamientos realizados es muy importante. Para agilizar las pruebas he confeccionado un modelo con bastantes menos parámetros que la ResNet50.

Este modelo además de varios parámetros menos, estamos hablando de que hay cuatro millones y medio casi de parámetros entrenables, está pensado para entrenar con imágenes 48x48 las cuales se procesan mucho más rápido. Por lo que los tiempos manejados con este nuevo modelo son significativamente menores.

La estructura de la red es la siguiente:

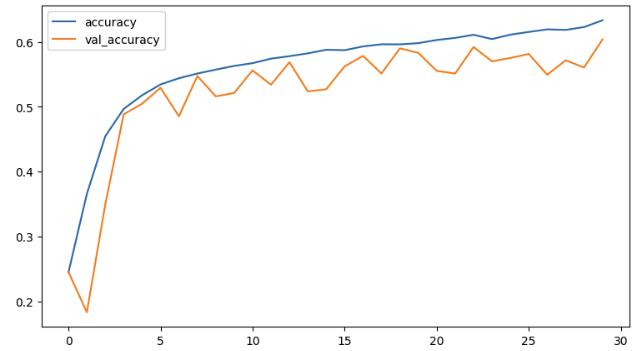
## Capítulo 4.7 - Data augmentation

Como hemos comentado antes en esta misma memoria es importante elegir bien las transformaciones que hacer para aumentar nuestro conjunto de datos, ya que si elegimos mal podemos acabar peor de lo que estábamos.

Para este problema las inversiones horizontales por ejemplo parecen una buena idea ya que da igual si una persona está mirando hacia su izquierda o su derecha para decir cual es su estado de ánimo.

Probemos entonces a aplicar ese aumento:

**FER-2013**



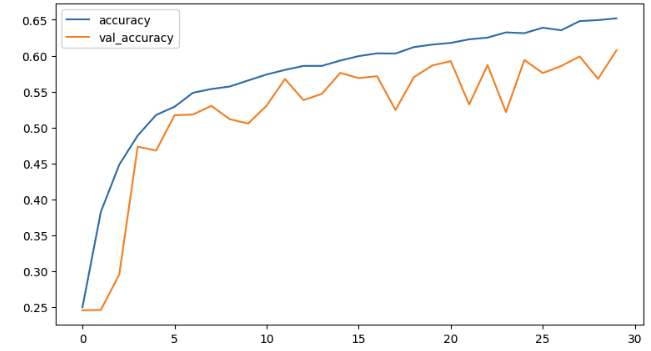
Claramente este cambio le ha venido muy bien al modelo, el accuracy en validación y entrenamiento se mantienen prácticamente iguales durante los 30 epochs.

Keras ofrece una transformación con la que deformar ligeramente la imagen.



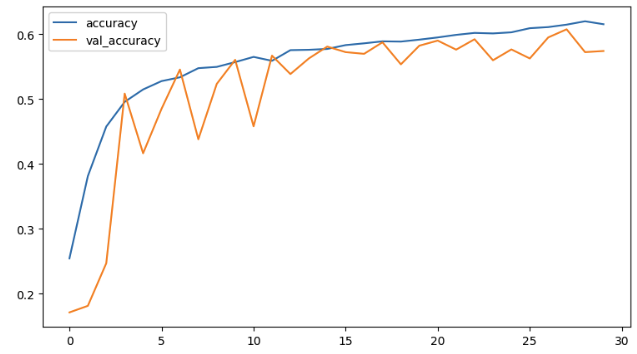
La imagen expuesta es una exageración para que se entienda la idea pero si lo hacemos más sutil el modelo va a pensar que son dos fotos distintas que es lo que buscamos, probemos.

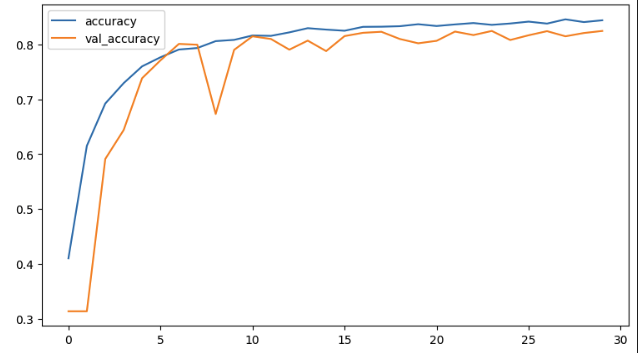
**FER-2013**



Esta transformación también ha sido positiva. ¿Y si mezclamos ambas?

**FER-2013**

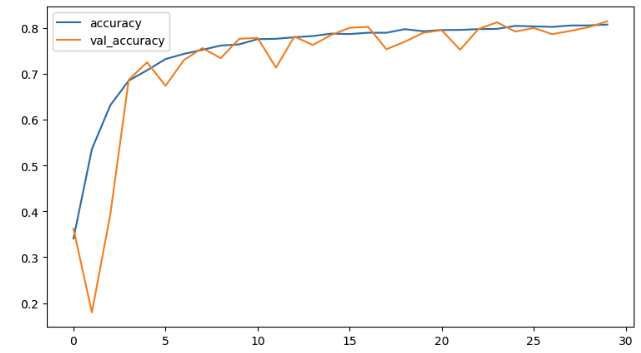


**FER-2013+**

Las dos se retroalimentan muy bien y tenemos un modelo con el que la diferencia entre las dos líneas es prácticamente inexistente. Todo esto sin tener que recoger más datos o hacer un gran cambio en la estructura de la red, todo muy bonito. ¿Pero y si hubiésemos elegido unas transformaciones malas?

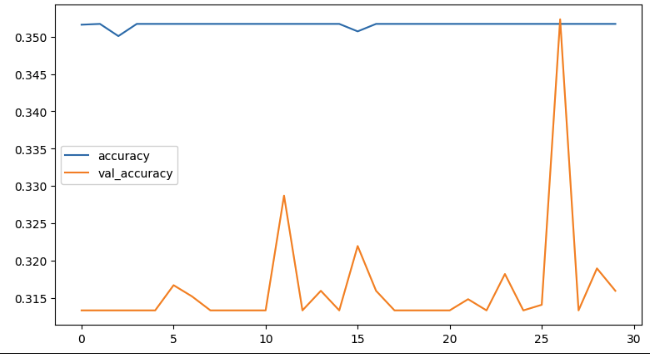
En este caso la inversión vertical, es decir que las personas aparezcan boca abajo, no tiene sentido. Veamos cuánto puede llegar a afectar negativamente este cambio junto con las transformaciones positivas anteriores.

**FER-2013+**



Sorprendentemente no tan mal como hubiésemos podido esperar, sin embargo cambiar levemente el brillo puede llegar a parecer una buena transformación. Así queda su combinación con inversión horizontal y las ligeras deformaciones:

**FER-2013+**



Terrible: esta transformación nos perjudica gravemente. Podemos sacar como conclusión que data augmentation es una gran forma de mejorar mucho nuestro modelo, pero hay que saber qué transformaciones escoger, lo cual es más difícil de lo que parece. La mejor forma es pensar lógicamente cuales pueden beneficiarnos, luego probarlas y quedarnos con las mejores. La intuición puede jugarnos una mala pasada como en el caso del brillo y las inversiones verticales.

## Capítulo 4.8 - Label smoothing

Uno de los grandes problemas de este conjunto de datos es la ambigüedad de las clases a las que pertenecen las imágenes, ya que rara vez tenemos emociones que pertenezcan sola y exclusivamente a una de las clases que tenemos, es por eso que es lógico pensar en usar label smoothing. Recapitulando: label smoothing nos permite evitar que el modelo tenga sobre confianza en sus predicciones y es justamente lo que buscamos.

Decidido, usemos label smoothing pero, ¿Qué usamos? Como con muchos hiperparametros en aprendizaje automático una de las formas más fáciles de elegir un buen valor es probar con varios.

| **FER-2013** | | |
| --- | --- | --- |
|  | accuracy/epoch | accuracy en test |
| 0.1 |  | 57.15% |
| 0.2 |  | 58.94% |
| 0.3 |  | 56.91% |

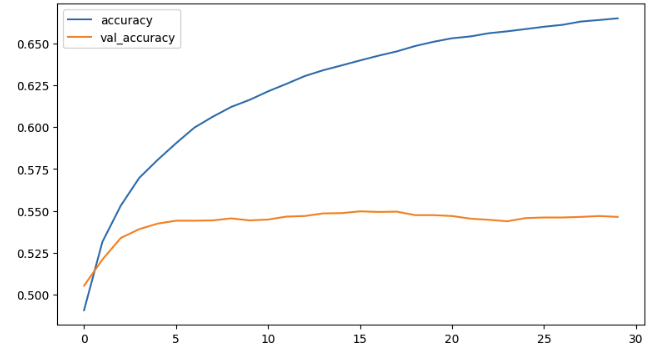
| **FER-2013+** | | |
| --- | --- | --- |
|  | accuracy/epoch | accuracy en test |
| 0.1 |  | 82.02% |
| 0.2 |  | 83.41% |
| 0.3 |  | 82.74% |
| 0.4 |  | 84.07% |

# Capítulo 5 - Resumen de resultados obtenidos

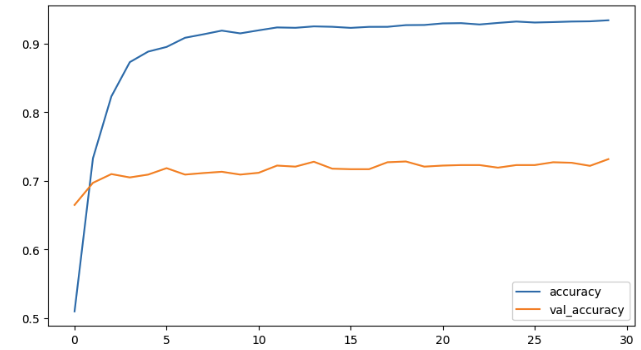
## Capítulo 5.1 - Transfer learning con una ResNet 50

|  | **Training** | **Validation** | **Test** | **Epochs** |
| --- | --- | --- | --- | --- |
| **Fer2013** | 67.74% | 54.64% | 54.68% | 30 |
| **Fer2013+** | 77.57% | 73.16% | 72.93% | 30 |

**FER-2013**

****

**FER-2013+**



En estas gráficas podemos ver que tanto el modelo de FER-2013 como el modelo de FER-2013+ sobre aprenden mucho, ya que el accuracy en validación y en entrenamiento se separan muy rápidamente.

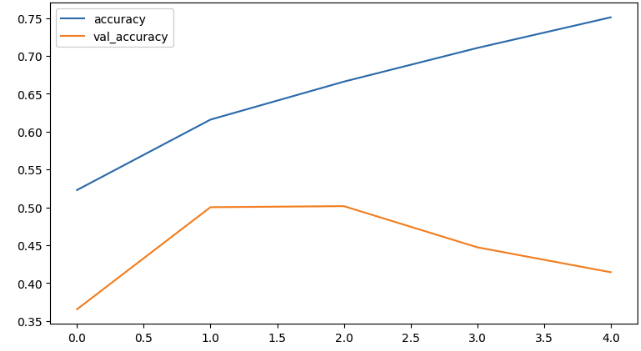
El eje de ordenadas es el accuracy y el de abcisas los epochs o lo que es lo mismo cuántas veces ha visto el modelo el conjunto de datos al completo

## 

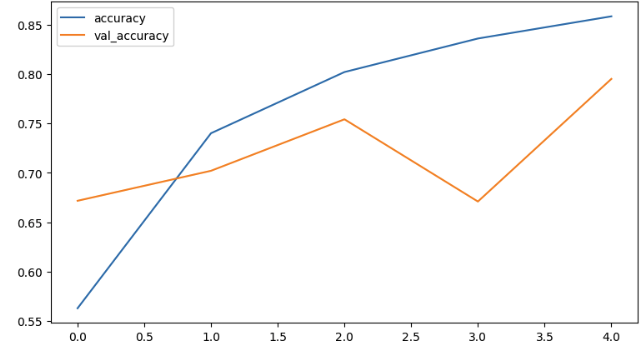
## Capítulo 5.2 - Fine tuning de la ResNet 50 anterior

|  | **Training** | **Validation** | **Test** | **Epochs** |
| --- | --- | --- | --- | --- |
| **Fer2013** | 52.28% | 41.44% | 42.1% | 5 |
| **Fer2013+** | 85.1% | 79.5% | 82.02% | 5 |

**FER-2013**



**FER-2013+**

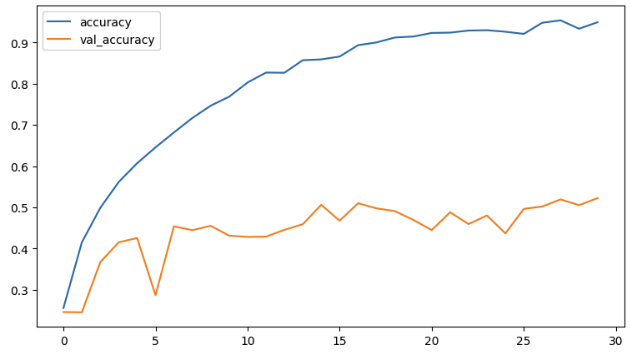


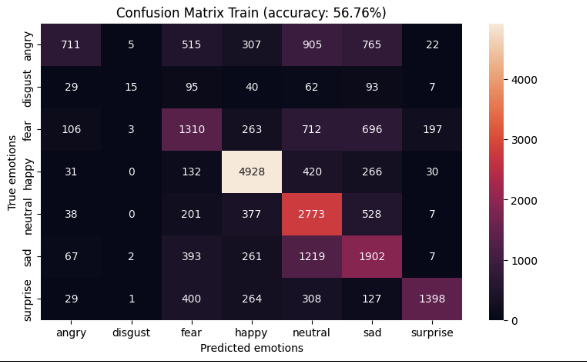
Debido a la alta demanda de recursos comentada solo he entrenado este modelo por 5 epochs, no es mucho y no se pueden sacar grandes conclusiones, pero se puede apreciar que el accuracy en validación no diverge tan rápido como antes del de entrenamiento. Esto da a pensar que con más epochs probablemente se obtendría mejor resultado.

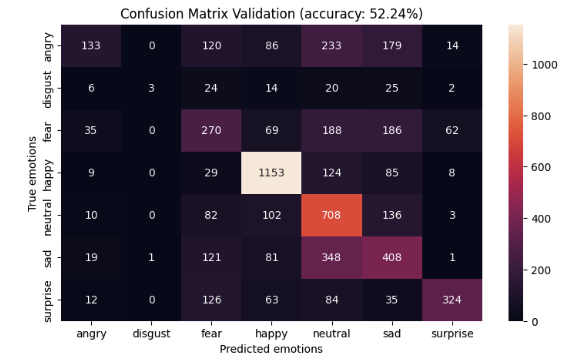
## Capítulo 5.3 - Usando una CNN personalizada sin modelo pre-entrenado

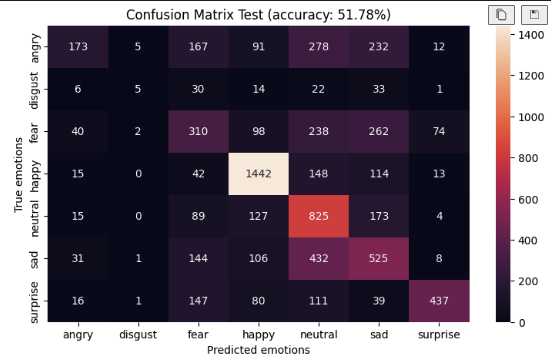
|  | **Training** | **Validation** | **Test** | **Epochs** |
| --- | --- | --- | --- | --- |
| **Fer2013** | 56.76% | 52.24% | 51.78% | 30 |
| **Fer2013+** | 81.6% | 77.47% | 78.1% | 30 |

**FER-2013**

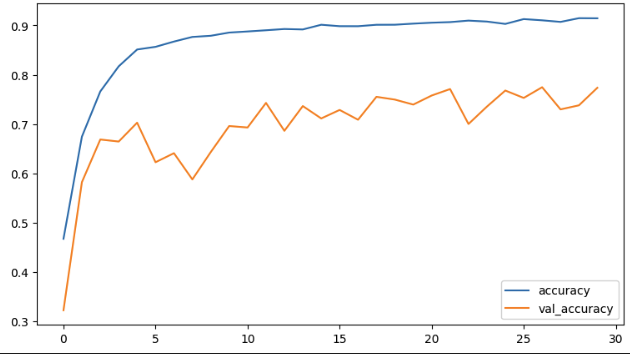


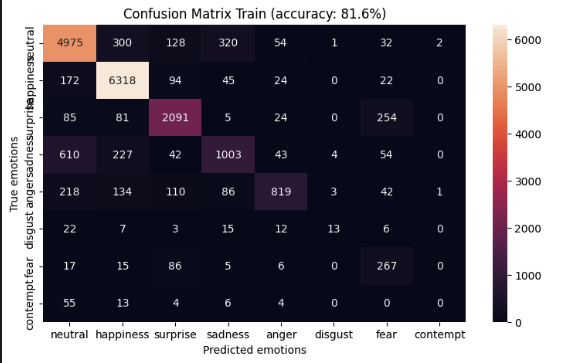


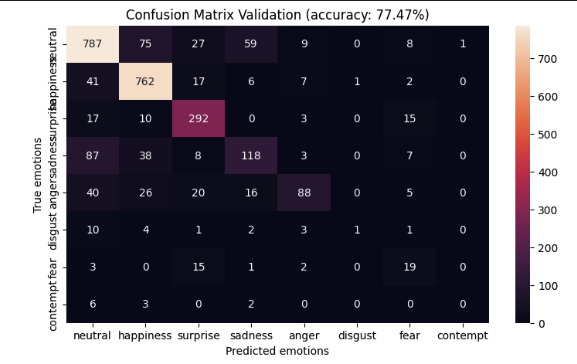


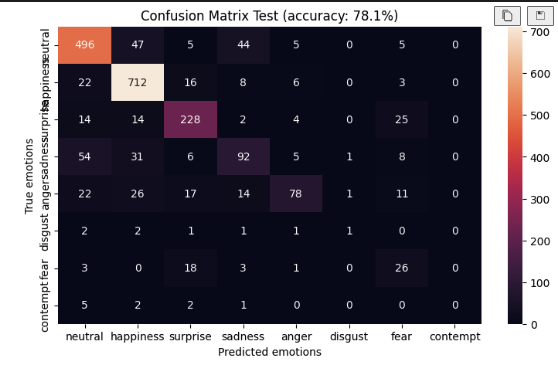


**FER-2013+**









Son modelos mejorables pero al mismo tiempo un buen punto de partida.

Estas son algunas de las características en las que se fija el modelo de FER-2013 para hacer sus predicciones, las que aparecen en rosa contribuyen positivamente a predecir la clase de esa fila, las que están en azul restan posibilidad de que la foto sea de esa clase. Están calculadas con shap values:



## 

## Capítulo 5.4 - Data augmentation

| **Fer2013** | | | | |
| --- | --- | --- | --- | --- |
|  | **Training** | **Validation** | **Test** | **Epochs** |
| **horizontal flip** | 65.63% | 60.36% | 59.6% | 30 |
| **shear range=5** | 70.41% | 60.74% | 60.67% | 30 |
| **horizontal flip y shear range=5** | 61.41% | 57.41% | 57.47% | 30 |
| **horizontal flip y shear range=10** | 62.61% | 58.81% | 58.89% | 30 |

| **Fer2013+** | | | | |
| --- | --- | --- | --- | --- |
|  | **Training** | **Validation** | **Test** | **Epochs** |
| **horizontal flip y shear range=5** | 85.65% | 82.46% | 82.83% | 30 |
| **horizontal flip y shear range=10** | 84.88% | 82.31% | 82.69% | 30 |
| **horizontal flip, shear range=5 y brightness\_range=(0.1,0.2)** | 35.49% | 31.6% | 36.39% | 30 |
| **horizontal flip, shear range=5 y vertical flip** | 82.79% | 81.52% | 81.35% | 30 |

## 

## Capítulo 5.5 - Label smoothing

| **Fer2013** | | | | |
| --- | --- | --- | --- | --- |
|  | **Training** | **Validation** | **Test** | **Epochs** |
| **0.1** | 61.34% | 57.45% | 57.15% | 30 |
| **0.2** | 63.69% | 59.76% | 58.94% | 30 |
| **0.3** | 60.87% | 57.08% | 56.91% | 30 |

| **Fer2013+** | | | | |
| --- | --- | --- | --- | --- |
|  | **Training** | **Validation** | **Test** | **Epochs** |
| **0.1** | 84.77% | 81.67% | 82.02% | 30 |
| **0.2** | 85.79% | 81.41% | 83.41% | 30 |
| **0.3** | 85.48% | 81.52% | 82.74% | 30 |
| **0.4** | 86.7% | 84.6% | 84.07% | 30 |

## Capítulo 5.6 - Test Time Augmentation

| **Fer2013** | |
| --- | --- |
| **Ejemplos generados para cada foto de test** | **Training** |
| **3** | 59.57% |
| **10** | 59.7% |
| **15** | 59.99% |
| **20** | 59.71% |

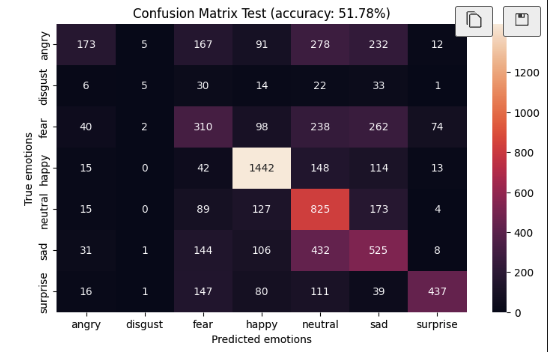
| **Fer2013+** | |
| --- | --- |
| **Ejemplos generados para cada foto de test** | **Training** |
| **3** | 84.98% |
| **5** | 85.17% |
| **10** | 84.84% |
| **20** | 85.27% |
| **25** | 85.32% |

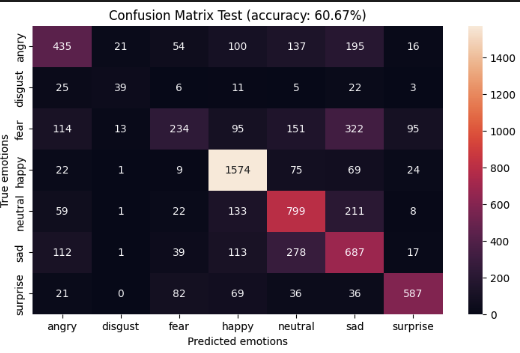
## 

# Capítulo 6 - Conclusiones y líneas futuras

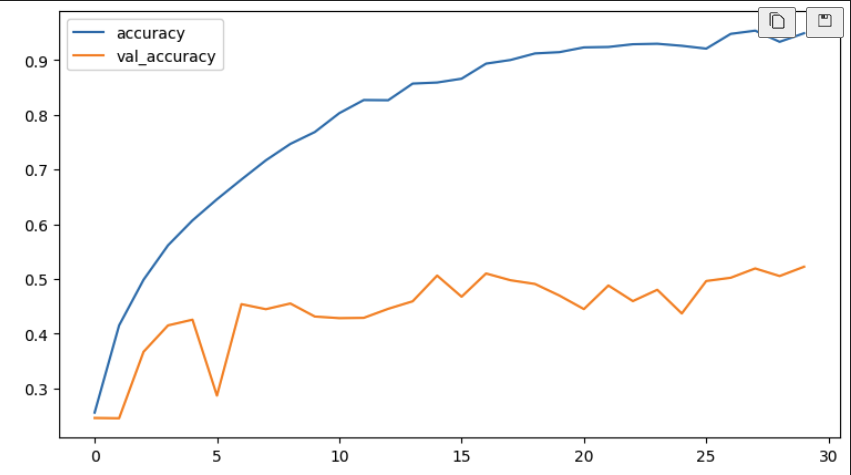
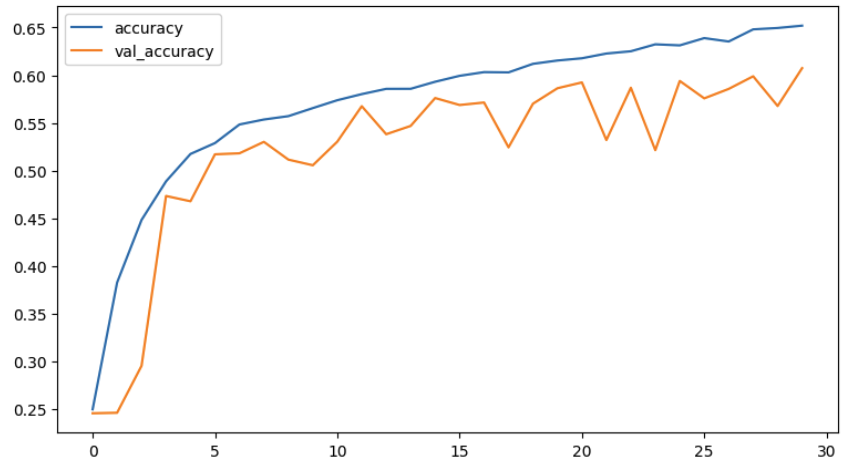
Empezamos con 51.78% de accuracy en test y terminamos con 60.67% en FER-2013.

En accuracy por clases hemos mejorado:



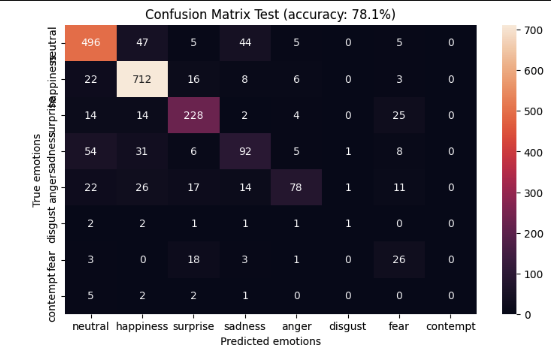


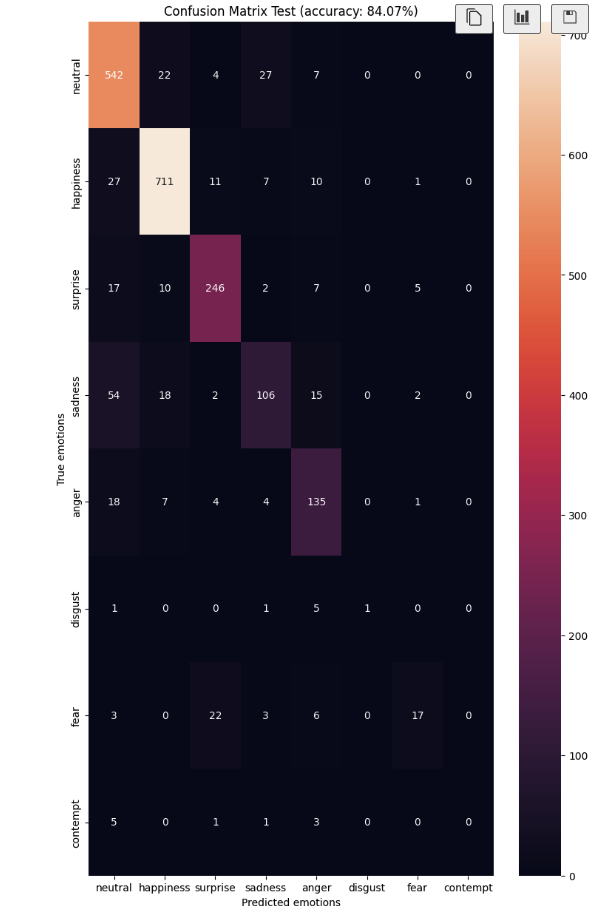
Tenemos además más margen de mejora ya que generaliza mucho mejor que al inicio.

 → 

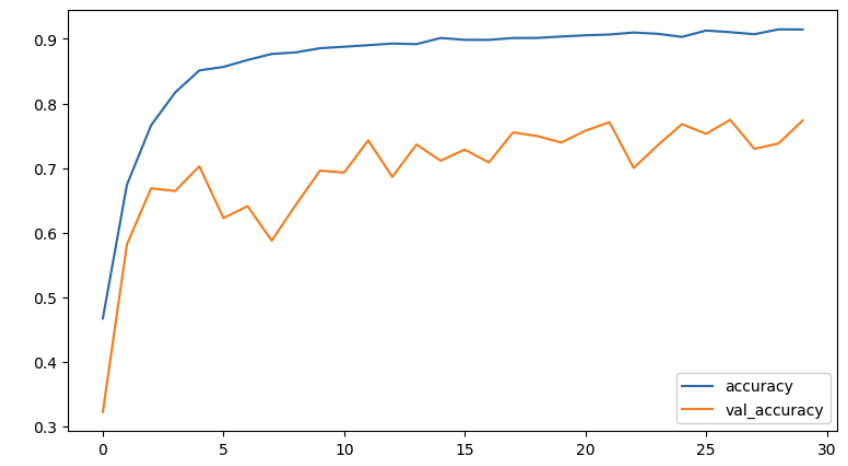
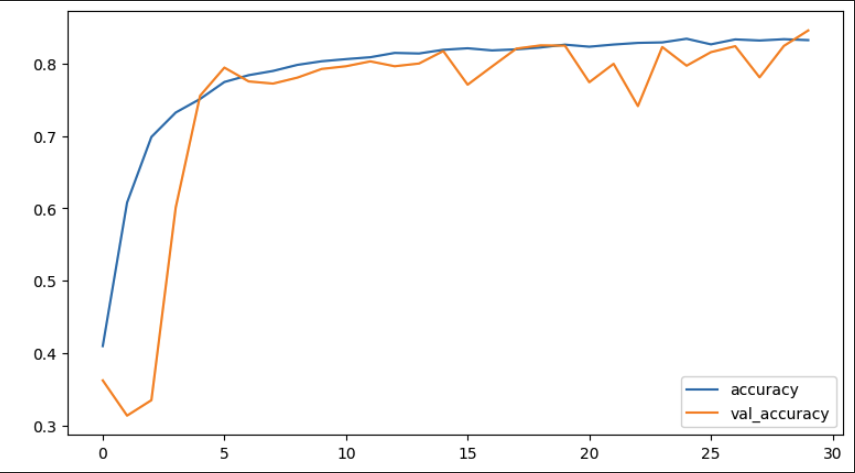
En FER-2013+ empezamos con 78.1% de accuracy en test y terminamos con un 85.32%.

En accuracy por clases también hemos mejorado:





Tenemos también más margen de mejora ya que generaliza mucho mejor que al inicio.

 → 

Las únicas técnicas no beneficiosas han sido el label smoothing y el test time augmentation en FER-2013, sin embargo en FER-2013+ nos han venido bien.

Por lo que creo que el balance es muy positivo, ya que la mejora es significativa y no hemos hecho cambios estructurales grandes, ni hemos tenido que invertir dinero y tiempo en aumentar el conjunto de datos.

Si tuviese más tiempo intentaría buscar más técnicas de mejora similares a las expuestas en este trabajo, probar con modelos base distintos y contrastar estas técnicas con más datasets además del usado.

# Anexos

[COMPARISON OF LEARNING ALGORITHMS FOR HANDWRITTEN DIGIT RECOGNITION](https://d1wqtxts1xzle7.cloudfront.net/48203614/KNN_recognition-libre.pdf?1471722187=&response-content-disposition=inline%3B+filename%3DComparison_of_Learning_Algorithms_for_Ha.pdf&Expires=1685679814&Signature=Jl32UQ26WqLptnBzvsX6EygEsE3YkR8kqSoWpNCNtM2teJWg6acMaELdgHtgCNt4UxLynzq8XwUB2Eljv-zZWQ6tN4IFf2RumL2EIRhMiPULmlWuVY07Q4UQ6PRBS19ayXog3VPL0YBGA8TrbxF31oqIW~9aM07Rx1BJCBzr1WuX7tNk94VandgZJK5QccLAqlzN4efDa6~PNkbX9xHzso5OMwf9y6lx~OdqNlT4IWA9QwVB0fRa7BdWqv1n62upXBWaNxJjShXspQ9VjIy8mC15wdjL~5~D16Vc0hn4zYasgit~GvDo5XtF7TV5Pt-sLGwZMBYGL2fAfwhN2VR38A__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZACOMPARISON%20OF%20LEARNING%20ALGORITHMS%20FOR%20HANDWRITTEN%20DIGIT%20RECOGNITION)

[J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848](https://ieeexplore.ieee.org/abstract/document/5206848)

[Russakovsky, O., Deng, J., Su, H. et al. ImageNet Large Scale Visual Recognition Challenge. Int J Comput Vis 115, 211–252 (2015).](https://doi.org/10.1007/s11263-015-0816-y)

[Very Deep Convolutional Networks for Large-Scale Image Recognition](https://doi.org/10.48550/arXiv.1409.1556)

[Stanko, I. (2020). The Architectures of Geoffrey Hinton. In: Skansi, S. (eds) Guide to Deep Learning Basics. Springer, Cham](https://doi.org/10.1007/978-3-030-37591-1_8)

[Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift](https://arxiv.org/abs/2006.06972)

[Repositorio con mi trabajo](https://github.com/alvarolarraya12/tfg)

[Dataset Fer2013](https://www.kaggle.com/datasets/msambare/fer2013)

[Dataset Fer2013+](https://github.com/microsoft/FERPlus)

[Algunas técnicas usadas](https://github.com/fastai/fastbook/blob/master/07_sizing_and_tta.ipynb)

[TTA keras](https://towardsdatascience.com/test-time-augmentation-tta-and-how-to-perform-it-with-keras-4ac19b67fb4d)

[Grid Search Keras](https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/)

[Shap values](https://github.com/slundberg/shap)

[Breve Historia de las Redes Neuronales](https://www.aprendemachinelearning.com/breve-historia-de-las-redes-neuronales-artificiales/)

[The Convolutional Classifier](https://www.kaggle.com/code/ryanholbrook/the-convolutional-classifier)

[Convolution and ReLU](https://www.kaggle.com/code/ryanholbrook/convolution-and-relu)

[MaxPooling](https://www.kaggle.com/code/ryanholbrook/maximum-pooling)

[The Sliding Window](https://www.kaggle.com/code/ryanholbrook/the-sliding-window)

[Data Augmentation](https://www.kaggle.com/code/ryanholbrook/data-augmentation)

[Dropout and Batch Normalization](https://www.kaggle.com/code/ryanholbrook/dropout-and-batch-normalization)

[ResNet50 architecture](https://datagen.tech/guides/computer-vision/resnet-50/)

[Transfer Learning y Fine-tuning](https://keepcoding.io/blog/transfer-learning-fine-tuning/)