Universitat Oberta de Catalunya

uoc.edu

Control de accesos con detección de mascarillas en tiempo real implementado con algoritmos de reconocimiento facial y Deep Learning

Ezequiel Cuadra
Ingeniería informática
Inteligencia Artificial

Nombre Consultor/a
Emilio Sansano Sansano

08/06/2021

Agradecimientos

Una etapa está llegando a su final y me gustaría aprovechar estas líneas para agradecer a todos los que estuvieron a mi lado estos años, a todos los que se han ido pero me siguen desde el cielo y a los que están en la distancia.

Agradecer a mi pareja Mónica, a mi familia, a mis amigos, a los becachondos, en especial a Javier que gracias a él empezó toda esta aventura, a todos los compañeros que he ido conociendo estos años y a todo el profesorado.

Gracias por todo y por tanto.

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Control de accesos con detección de mascarillas en tiempo real implementado con algoritmos de reconocimiento facial y Deep Learning*
Nombre del autor:	*Ezequiel Nicolás Cuadra*
Nombre del consultor/a:	*Emilio Sansano Sansano*
Nombre del PRA:	*Joan Arnedo Moreno*
Fecha de entrega (mm/aaaa):	08/2021
Titulación::	*Grado de Ingeniería Informática*
Área del Trabajo Final:	*Inteligencia Artificial*
Idioma del trabajo:	*Español*
Palabras clave	*Control de accesos, detección de mascarillas, reconocimiento facial*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

Debido a la pandemia que estamos sufriendo a causa de la covid-19, los gobiernos han tenido que adoptar una serie de restricciones y medidas de prevención para evitar la propagación de dicho virus.

La finalidad de este proyecto es crear un sistema de control de accesos con detección de mascarillas en tiempo real, implementado con algoritmos de *Deep Learning* diseñados para la visión por computador.

A través de técnicas de *Transfer Learning* vamos a aprovechar una red pre entrenada con un gran poder computacional y sobre un enorme *dataset* de imágenes (*Imagenet*). Congelaremos las primeras capas y crearemos una red que se acople a la cabeza de este modelo para entrenar con nuestros datos.

Este sistema puede ser utilizado en diferentes entornos para asegurar que las medidas de seguridad se están cumpliendo correctamente, como pueden ser fábricas, transporte público, empresas y más.

Se concluye que una red previamente entrenada en reconocimiento de imágenes es capaz de reconocer caras y clasificar adecuadamente si lleva

mascarilla, si se lleva bien puesta o si por otra parte no se lleva. Esto se consigue con el entreno y acoplamiento de nuestra red con el modelo previamente entrenado.

El desarrollo del trabajo se llevará a cabo en lenguaje Python, con bibliotecas de apoyo como Keras para las redes neuronales, Pandas para la manipulación de datos, Numpy para funciones matemáticas, OpenCV para la visión por computador, entre otras.

Abstract (in English, 250 words or less):

Due to the pandemic we are suffering because of covid-19, governments have had to adopt a series of restrictions and preventive measures to avoid the spread of this virus.

The purpose of this project is to create an access control system with real-time mask detection, implemented with Deep Learning algorithms designed for computer vision.

Through Transfer Learning techniques we will take advantage of a pre-trained network with a great computational power and on a huge image dataset (Imagenet). We will freeze the first layers and create a network that will be coupled to the head of this model to train with our data.

This system can be used in different environments to ensure that security measures are being properly enforced, such as factories, public transportation, enterprises and more.

It is concluded that a network previously trained in image recognition is able to recognize faces and classify properly if a mask is worn, if it is worn correctly or if it is not worn at all. This is achieved by training and coupling our network with the previously trained model.

The development of the project will be carried out in Python language, with support libraries such as Keras for neural networks, Pandas for data manipulation, Numpy for mathematical functions, OpenCV for computer vision, and others.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Estamos en una situación excepcional debido a la pandemia ocasionada por el COVID-19, este TFG se centra en una de las tres medidas preventivas vigentes más populares para evitar la propagación del virus, la obligatoriedad de portar mascarilla en espacios públicos o cerrados.

En este trabajo se pretende desarrollar una solución para el control de accesos (como podrían ser establecimientos comerciales, áreas de trabajo, centros de reuniones, fábricas, etc.), implementada con algoritmos de reconocimiento facial y *Deep Learning* para detectar la portabilidad y el correcto uso de las mascarillas.

A medida que el gobierno va relajando las restricciones impuestas con el objetivo de evitar la transmisión del virus, las personas tienden también a adoptar una actitud más distendida frente al virus. Esto lo podemos ver por ejemplo, en las grandes aglomeraciones de personas que se amontonan para irse de fiesta y la falta de distanciamiento que ello conlleva o la ausencia de mascarillas, personas que simplemente por descuido no llevan correctamente la mascarilla cuando acceden a recintos comerciales, etc.

Hasta que no se consiga una vacunación masiva, el uso de mascarillas reduce considerablemente el índice de contagio. Con este trabajo se pretende dotar de una herramienta que ayude a evitar la propagación del virus.

1.2 Objetivos del Trabajo

Tenemos unos objetivos muy claros con respecto a este trabajo, son los siguientes:

- Que el modelo diseñado sea capaz de detectar y alertar cuando una persona no lleve correctamente la mascarilla
- Aprovechar un modelo pre entrenado para conseguir la detección
- Obtener unos resultados óptimos

1.3 Enfoque y método seguido

La estrategia es adaptar un producto ya existente, como puede ser el siguiente:

[1] https://github.com/balajisrinivas/Face-Mask-Detection

Para acondicionarlo a nuestras necesidades como pueden ser, ajustarlo a nuestro conjunto de datos, que vamos a crear a partir de otros ya existentes para intentar aumentar el aprendizaje del modelo o añadir una alarma sonora cuando se detecte que no se lleva mascarilla. Ajustar los parámetros para conseguir un buen rendimiento.

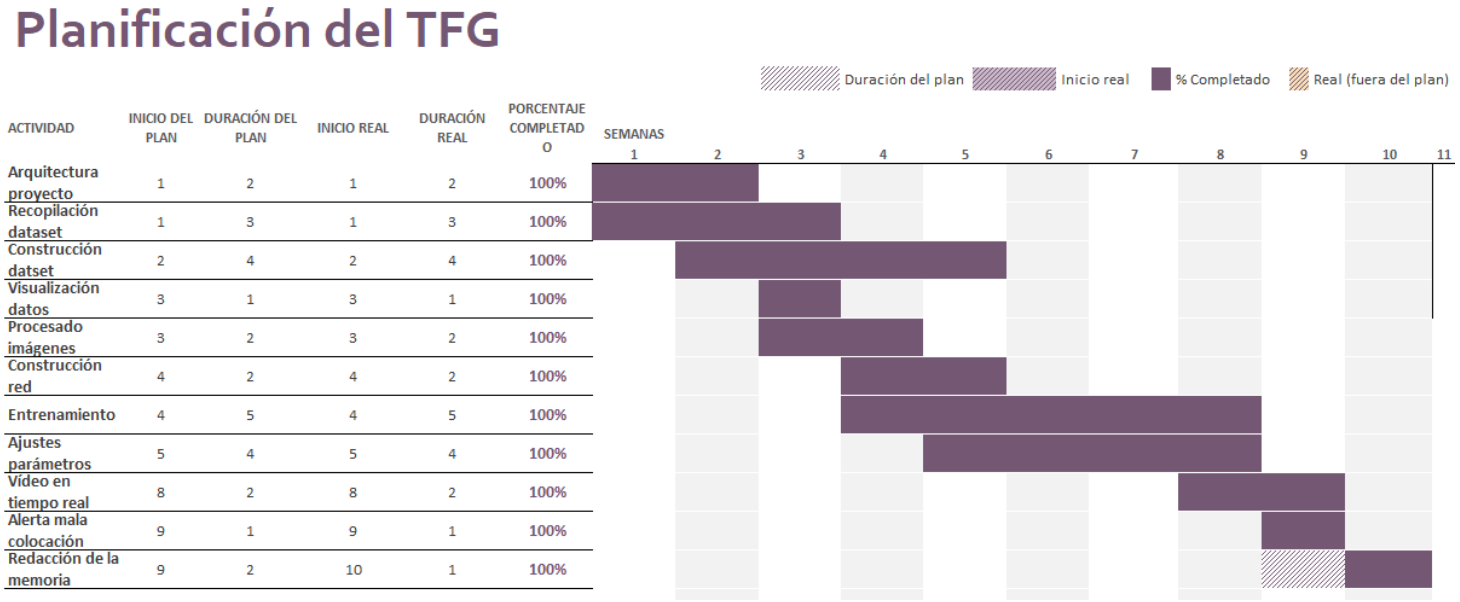
Para conseguir nuestros objetivos, aplicaremos como se ha dicho anteriormente, la potencia de los modelos previamente entrenados para aprovechar todo ese poder computacional que se ha invertido y reaprovecharlo para nuestro propósito.

1.4 Planificación del Trabajo

Como recursos necesitaremos básicamente un equipo con Python, un *dataset* preparado para alimentar a la red y opcionalmente una tarjeta gráfica para que el entrenamiento sea más rápido.

Las tareas que llevaremos a cabo serán las siguientes:

- Preparación de la arquitectura del proyecto
- Recopilación de *datasets*
- Construcción de un *dataset* adecuado (combinando más de uno)
- Visualización de los datos
- Procesado de imágenes
- Construcción de nuestra red, la cual acoplaremos a la pre entrenada
- Entrenamiento de las últimas capas de la red
- Ajustes en los parámetros
- Desarrollo para capturar vídeo en tiempo real
- Implementación de una alerta como aviso de una mala colocación de la mascarilla



ACTIVIDAD	INICIO DEL PLAN	DURACIÓN DEL PLAN	INICIO REAL	DURACIÓN REAL	PORCENTAJE COMPLETADO
Arquitectura proyecto	1	2	1	2	100%
Recopilación dataset	1	3	1	3	100%
Construcción datset	2	4	2	4	100%
Visualización datos	3	1	3	1	100%
Procesado imágenes	3	2	3	2	100%
Construcción red	4	2	4	2	100%
Entrenamiento	4	5	4	5	100%
Ajustes parámetros	5	4	5	4	100%
Vídeo en tiempo real	8	2	8	2	100%
Alerta mala colocación	9	1	9	1	100%
Redacción de la memoria	9	2	10	1	100%

Imagen 1: Planificación TFG

1.5 Breve sumario de productos obtenidos

Al final del trabajo, obtendremos un programa que capturará vídeo a través de una *webcam* y detectará si la gente lleva o no mascarilla, emitiendo una alerta sonora en caso de no detectarla.

1.6 Breve descripción de los otros capítulos de la memoria

En el capítulo 2. ¿Qué es la visión por computador?, se hará una introducción de qué es y cómo trabaja una red convolucional, qué son los modelos pre entrenados y sus beneficios, por último hablaremos sobre la técnica de *Transfer Learning*.

En el capítulo 3. Conjunto de datos, se presentarán los datos que se han utilizado para este trabajo, se analizarán y visualizarán los datos (se graficará la distribución entre las clases, número de imágenes de cada clase, etc.) y se hablará sobre la preparación de las imágenes para poder entrenar el modelo

En el capítulo 4. Entreno y validación del modelo, se verá cómo ha ido el entreno y se presentarán datos de validación (matriz de confusión, métricas, etc.)

En el capítulo 5. Pruebas con vídeo en tiempo real, se adjuntará un vídeo probando el modelo

El resto de capítulos estarán formados por las conclusiones, glosario, bibliografía y anexos

2. ¿Qué es la visión por computador?

La visión artificial o visión por computador es una rama de la inteligencia artificial la cual permite a las máquinas “tener ojos” y poder ver el entorno que las rodea. Esta disciplina aglomera diferentes conjuntos de herramientas y métodos que permiten adquirir, procesar, analizar y comprender las imágenes del mundo real.

En la última década los avances dentro de este campo han sido absolutamente notables. Todo empezaba con la red neuronal bautizada como LeNet-5 creada por Yann LeCun en 1998, un sistema capaz de reconocer dígitos escritos a mano digitalizados en imágenes de 32x32 píxeles.

Lo que ha hecho que la visión por computador haya revolucionado tanto el sector tecnológico en estos últimos años, ha sido la fusión de estos algoritmos con el aprendizaje automático. Esto permite crear sistemas capaces de clasificar imágenes, detectar objetos, conducción autónoma de vehículos, entre muchos otros.

Así como estos sistemas son capaces de ser utilizados con distintas finalidades, también son amplios los sectores donde se aplican, como por ejemplo en: medicina, agricultura, banca, automoción, logística y un largo listado de industrias.

2.1 ¿Qué es y cómo trabaja una red neuronal convolucional?

En el apartado anterior comentábamos que los sistemas de visión por computador habían tenido su auge gracias a la combinación de estos algoritmos, con el aprendizaje automático. Ahora bien, este gran avance se debe en su mayoría a la aplicación de redes neuronales convolucionales.

Pero, ¿qué es una red neuronal convolucional? Las redes neuronales convolucionales (*Convolutional Neural Network en inglés*) CNN de aquí

en adelante, son un tipo de red neuronal artificial que fueron diseñadas para imitar el córtex visual del ojo.

La importancia de estas redes viene de su capacidad de poder descifrar los patrones más complejos en enormes *datasets* de imágenes, dotando de ojos a máquinas que tanto pueden observar rostros de personas, como radiografías de pacientes, como los peatones que se cruzan ante un coche autónomo.

Para entender un poco el funcionamiento de las CNN, vamos a analizar esta imagen:

Imagen 2: Ojo

¿Cómo sabemos que esto es un ojo? Bueno, porque hemos detectado una serie de elementos que conforman normalmente a un ojo: una pupila negra, líneas que son pestañas, superficies blancas y todo esto también lo hemos sabido reconocer porque somos capaces de detectar patrones circulares, cambios de contrastes, texturas y así sucesivamente.

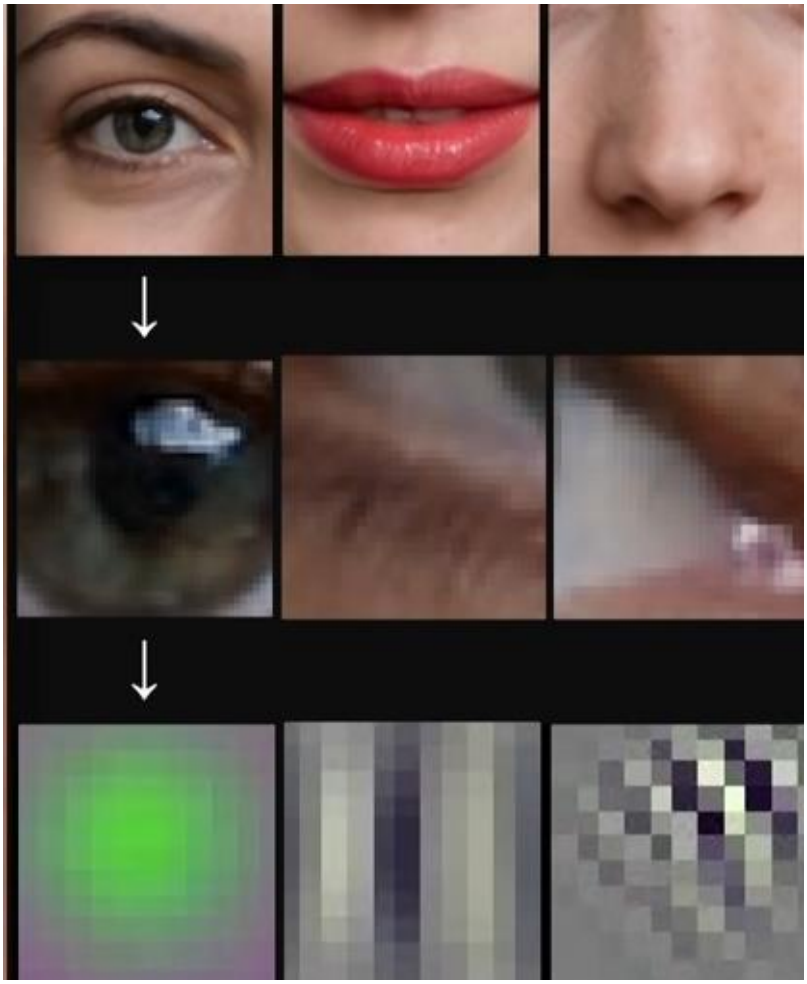


Imagen 3: Ojo en patrones

Al final si reproducimos los pasos que realiza nuestro córtex visual, con lo que nos encontraremos es con un procesamiento en cascada donde primero se identifican aquellos elementos básicos y generales y donde, en posteriores capas, esto se combina para generar patrones cada vez más complejos

Con esta primera aproximación de cómo funciona una CNN, vamos a explicar un poco su arquitectura y ver cuáles son las capas involucradas en este procesamiento en cascada que permite identificar patrones.

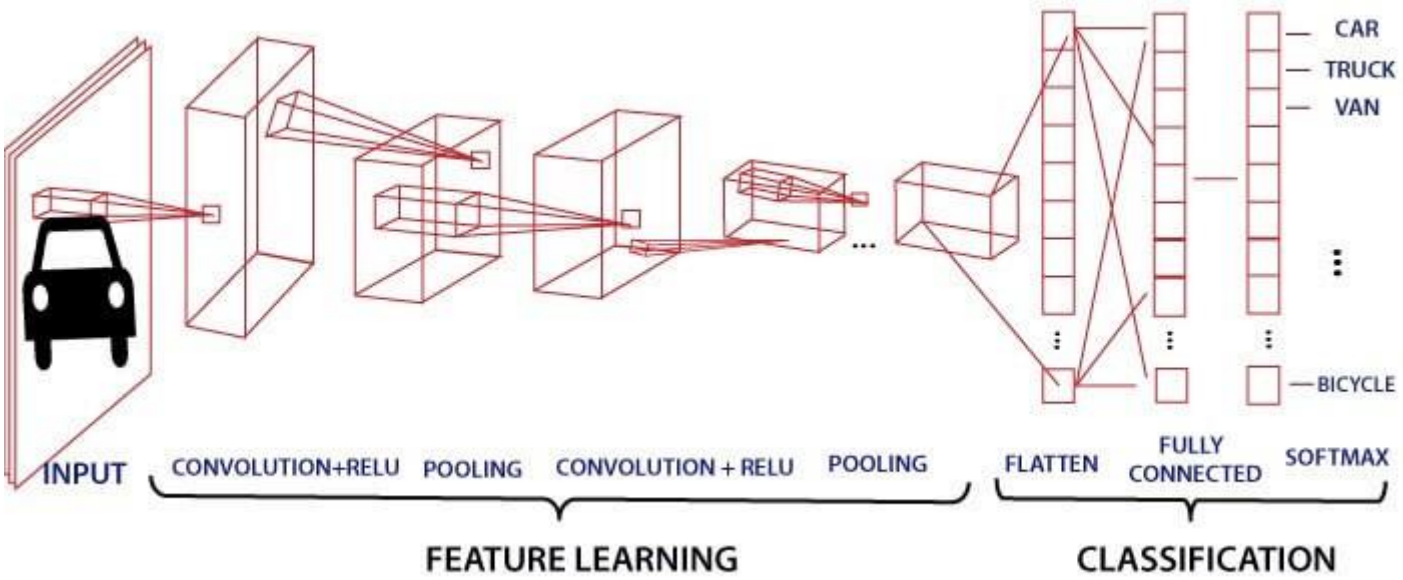


Imagen 4: Arquitectura de una CNN. Fuente: https://ichi.pro/es/red-neuronal-convolucional-cnn-y-su-aplicacion-todo-lo-que-necesita-saber-266752320713701

Como se puede observar partimos de un *input*, es decir, aquella imagen que queremos analizar.

Las CNN se caracterizan por contener un tipo de capa donde se realiza una operación matemática conocida como convolución.

Una convolución aplicada sobre una imagen, no es más que una operación que jugando con los valores de los píxeles es capaz de producir una nueva imagen. Cada pixel nuevo que se vaya a generar se calculará colocando una matriz de números llamados filtros, sobre la imagen original y donde se multiplicarán y sumarán los valores de cada pixel vecino para obtener así el nuevo valor. Siempre acompañada de una función de activación, que es una función que transmite la

información por las conexiones de las capas (en este ejemplo hay una función de activación ReLU, Rectificador Lineal Unitario).

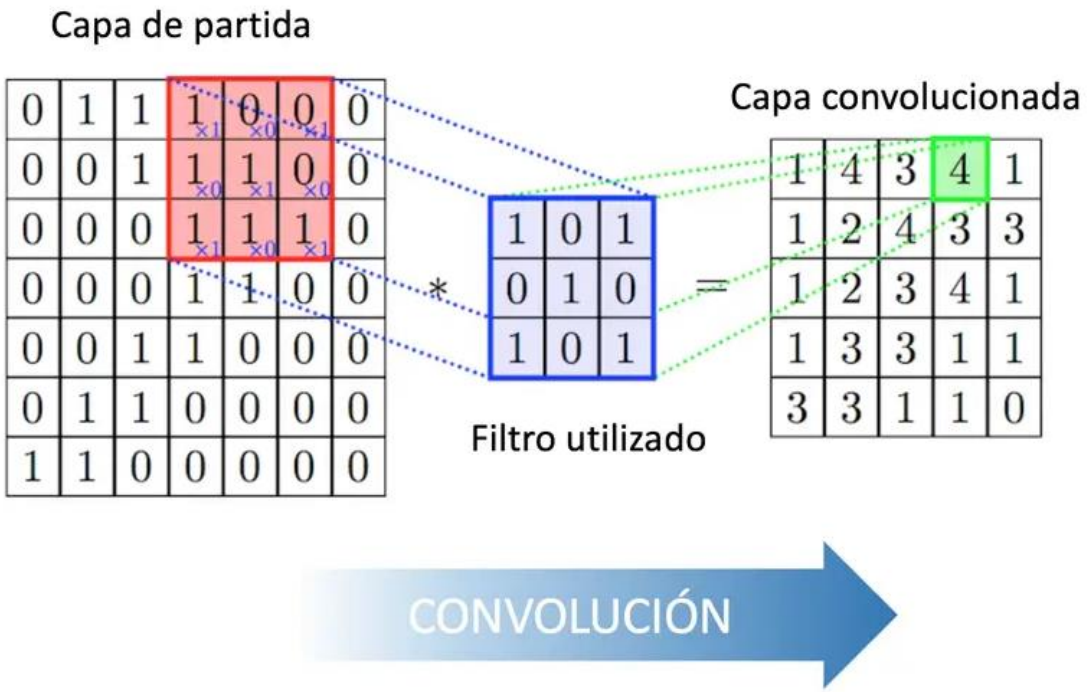


Imagen 5: Filtro de convolución. Fuente: https://www.diegocalvo.es/red-neuronal-convolucional/

Seguidamente tenemos una capa de *pooling* o de reducción, esta capa generalmente se coloca después de la capa de convolución. Es muy recomendable utilizarla, ya que reduce las dimensiones espaciales de la entrada para la siguiente capa convolucional. No obstante, siempre que se realiza una reducción de la muestra viene implícita en ella la pérdida de información.

Existen diferentes métodos para aplicar en una capa de reducción, veamos dos ejemplos:

- maxPooling: encuentra el valor máximo entre una ventana de muestra y pasa este valor como resumen de características sobre esa área

- avgPooling: encuentra la media entre una ventana de muestra y pasa este valor como resumen de características sobre esa área

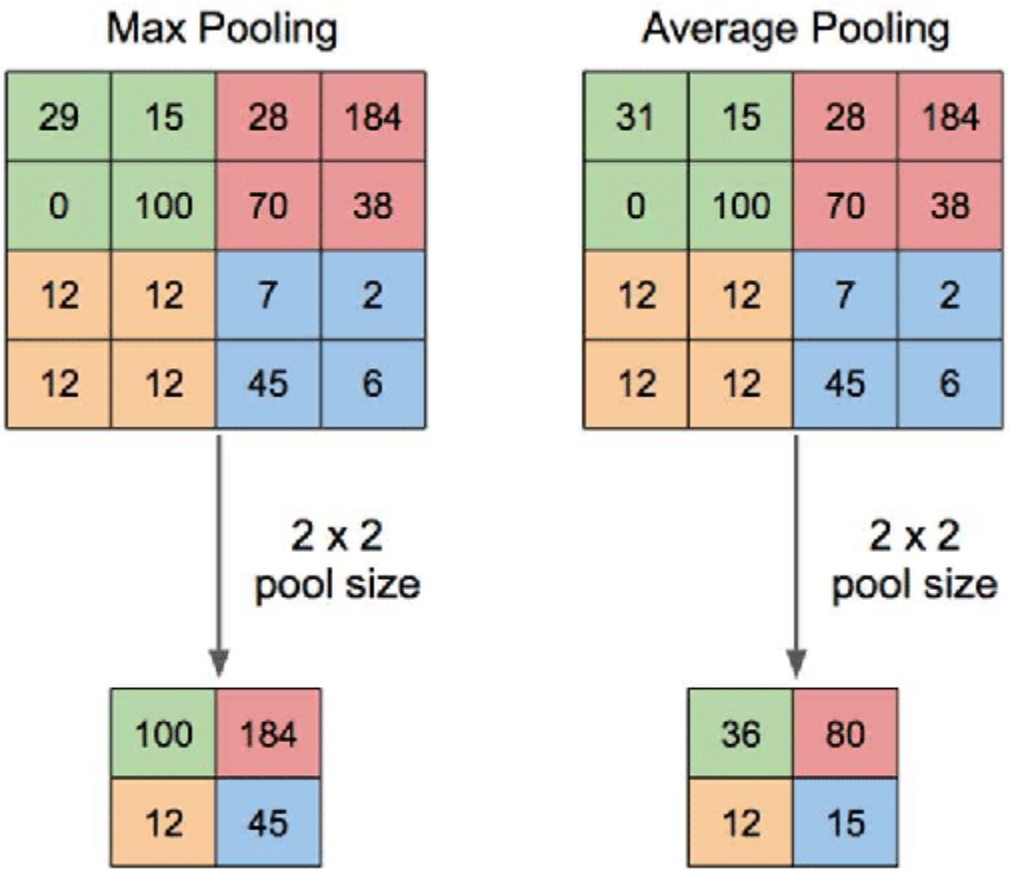


Imagen 6: Comparación entre maxPooling y avgPooling. Fuente: https://www.researchgate.net/figure/Illustration-of-Max-Pooling-and-Average-Pooling-Figure-2-above-shows-an-example-of-max_fig2_333593451

Como observamos en la imagen 3, se vuelve a aplicar una capa de convolución con otra de *pooling*, se pueden añadir cuantas capas de convolución se crean necesarias. Sin olvidar que cuanto más reduzcamos las muestras, más información perderemos en el camino.

Hasta el momento hemos definido la parte de la arquitectura encargada de la extracción de características de la imagen. Vamos a pasar a la parte de la red encargada de la clasificación o la regresión, dependiendo de la finalidad con la que entrenemos a la red.

Cuando hayamos llegado a nuestra última capa de reducción, tendremos una matriz con toda la información que se ha ido comprimiendo durante las capas anteriores. Es aquí donde entra en juego el aprendizaje automático.

Para ello deberemos convertir la matriz en un vector plano, para que pueda ser consumido por las neuronas de la red neuronal. Este proceso se conoce como aplanamiento o *flattening*.

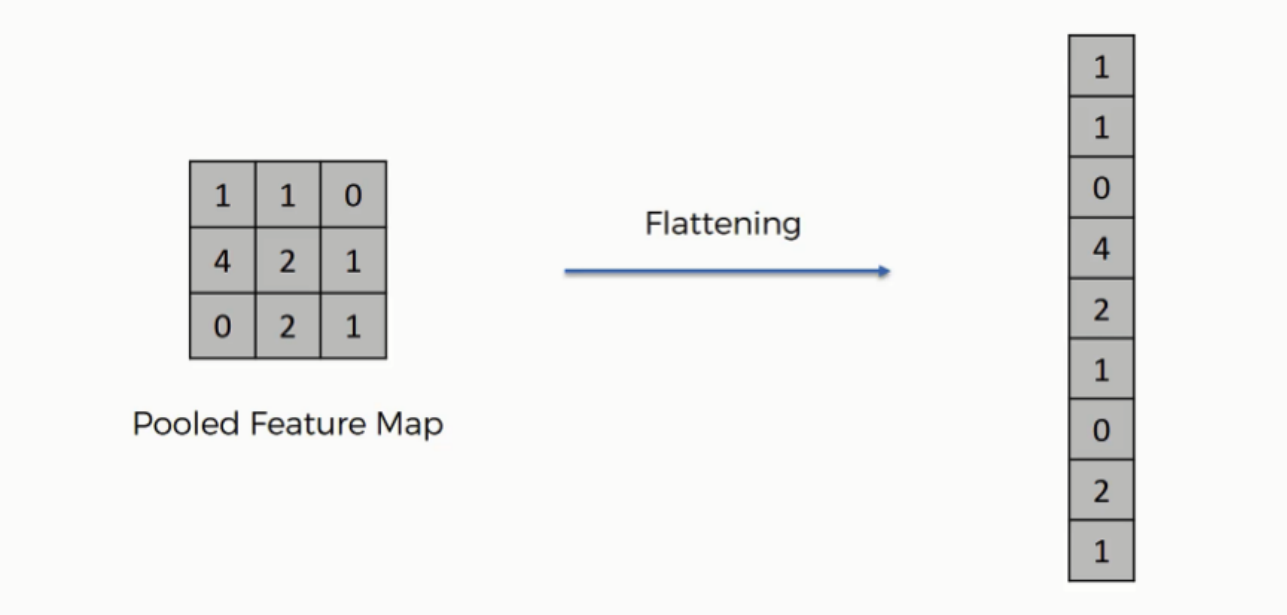


Imagen 7: Proceso de aplanamiento de una matriz. Fuente: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening

La dimensión de este vector será el tamaño de la primera capa de la red neuronal, siguiendo el ejemplo de la imagen 3, tenemos una red neuronal totalmente conectada. Nuevamente, podemos añadir cuantas capas creamos oportuno. Algo muy utilizado para evitar que un modelo se sobreajuste o desajuste (*overfitting y underfitting* en inglés) a las muestras en las fases de entrenamiento, es añadir capas de *dropout* que se encargaran de desactivar aleatoriamente un porcentaje de neuronas, ya que las neuronas cercanas entre sí suelen aprender patrones que se

relacionan y estas relaciones pueden llegar a formar un patrón muy específico con los datos de entrenamiento.

Para acabar con la arquitectura de la CNN nos falta añadir una última capa, la encargada de clasificar o predecir nuestro objetivo. Esta capa tendrá una función de activación diferente dependiendo de cuál sea nuestro cometido, por ejemplo:

- Softmax: es una función que se utiliza para representar una distribución categórica, es decir, cual es la probabilidad de que nuestra imagen pertenezca a cierta clase objetivo.

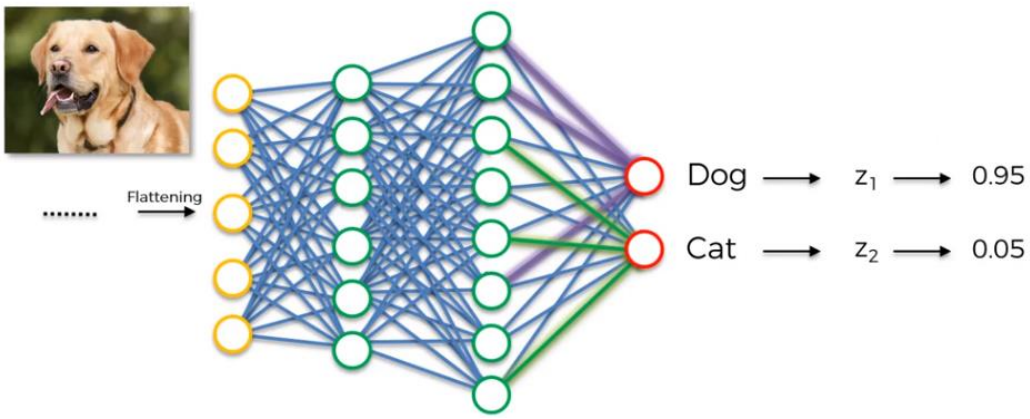


Imagen 8: Función softmax para predicción entre dos clases, gato o perro. Fuente: https://www.andreaperlato.com/aipost/cnn-and-softmax/

- ReLU: en cambio si lo que queremos es predecir un número, como podría ser el precio de algún producto, o el peso de una persona, necesitamos aplicar la función ReLU

2.2 Modelos pre entrenados

El concepto de modelo pre entrenado es simple, se trata de un modelo que ha sido creado por otras personas o entidades y entrenado con un ingente conjunto de datos de referencia y una gran capacidad de computación para resolver un problema específico.

Por ejemplo si se quiere construir un coche autónomo, una de las características importantes que necesita implementar es que sea capaz de reconocer el entorno que le rodea. Se pueden dedicar años a construir un algoritmo de reconocimiento de imágenes decente desde cero o se puede empezar a trabajar sobre un modelo pre entrenado por Google (GoogLeNet o también conocido como Inception V1) que se construyó sobre los datos de ImageNet para identificar las imágenes de esas fotos.

Una vez esta red se ha construido y se ha entrenado con un enorme *dataset*, para que otras personas puedan aprovechar todo este conocimiento adquirido a lo largo del proceso, se tienen que guardar los pesos de las neuronas.

El peso de una neurona es un valor ponderado que en un primer momento se inicializa como un valor aleatorio. Para que la red sea capaz de aprender, se utilizan las llamadas funciones de pérdidas que se encargan de minimizar el error en los resultados que estamos evaluando.

En la fase de entreno, los pesos de las neuronas se van actualizando con la finalidad de reducir el valor de dicha función de pérdida. Esto se consigue con la propagación hacia atrás (*backpropagation)*, que consiste en propagar los valores de error desde la capa de salida hacia todas las neuronas de las capas ocultas y repetir el proceso hasta llegar a un valor óptimo.

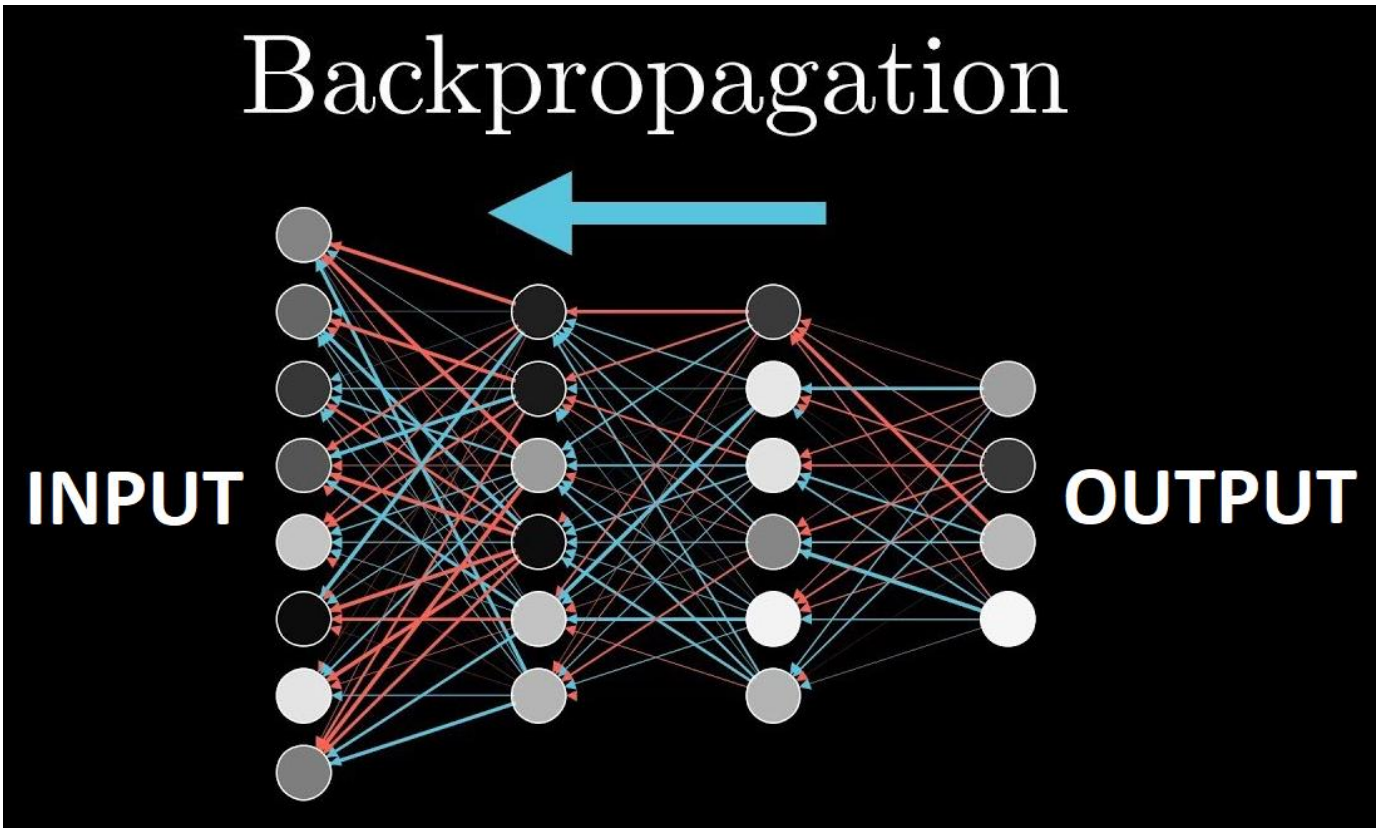


Imagen 9: Backpropagation. Fuente: https://medium.com/@sallyrobotics.blog/backpropagation-and-its-alternatives-c09d306aae4c

Veamos algunos de los modelos pre entrenados que tiene en su repositorio Keras:

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	-	-	5,330,571	-
EfficientNetB1	31 MB	-	-	7,856,239	-
EfficientNetB2	36 MB	-	-	9,177,569	-
EfficientNetB3	48 MB	-	-	12,320,535	-
EfficientNetB4	75 MB	-	-	19,466,823	-
EfficientNetB5	118 MB	-	-	30,562,527	-
EfficientNetB6	166 MB	-	-	43,265,143	-
EfficientNetB7	256 MB	-	-	66,658,687	-

Imagen 10: Modelos pre entrenados disponibles en Keras. Fuente: https://keras.io/api/applications/

En la tabla observamos diferentes características de cada modelo como pueden ser:

- Tamaño, precisión, cantidad de parámetros y la profundidad (cantidad de capas ocultas)

2.3 *Transfer Learning*

Cuando hayamos decidido implementar el modelo pre entrenado que más se adapte a nuestro objetivo final, es el momento de transferir todo ese conocimiento a nuestro dominio. Esto se lleva a cabo con la técnica de aprendizaje por transferencia (*transfer learning*).

El *transfer learning* tiene el objetivo de inferir conocimiento a partir de un conjunto de ejemplos de entrenamiento especificados para una tarea. Es decir, el aprendizaje previo realizado por el modelo nos permite utilizarlo en nuevas tareas que estén relacionadas.

La clave de esta técnica viene dada por la capacidad que tenga el modelo para generalizar el conocimiento para una nueva tarea. Podemos ver un claro ejemplo en la visión por computador, las características de bajo nivel como pueden ser formas, esquinas, líneas y demás, se pueden compartir tanto para una tarea de clasificación de objetos como para una de reconocimiento facial.

Hay varias técnicas para personalizar un modelo pre entrenado, vamos a ver dos de ellas:

- Extracción de características: Utilizar el conocimiento aprendido por una red anterior para extraer características significativas de las nuevas muestras, como comentábamos anteriormente pueden ser formas, líneas, etc.

  Simplemente se añade un nuevo clasificador, que será entrenado desde cero, sobre el modelo pre entrenado para poder reutilizar los mapas de características aprendidos previamente para el conjunto de datos, es decir, se reemplazará la capa de salida de la red para conectar la última capa oculta con nuestra capa de entrada.

No será necesario re entrenar todo el modelo, la red convolucional base ya contiene características que son genéricamente útiles.

Sin embargo, la parte final de clasificación del modelo pre entrenado es específica para la tarea original y, por tanto, específica para el conjunto de clases en el que se ha entrenado el modelo. Es por ello que tendremos que “congelar” las capas del modelo base (para mantener los pesos de las neuronas encargadas de detectar patrones básicos) y entrenar sobre nuestras capas que serán las encargadas de hacer la nueva clasificación.

- Ajuste fino o *fine-tuning*: Consiste en descongelar algunas de las capas superiores de un modelo base congelado y entrenar conjuntamente las capas de clasificación recién añadidas y las últimas capas del modelo base. Esto nos permite "afinar" los mapas de características de orden superior en el modelo base para hacerlas más relevantes para la tarea específica que queramos llevar a cabo.

3. Conjunto de datos

Después de exponer un poco sobre la teoría que envuelve a las redes convolucionales, vamos a entrar en la parte más práctica de este proyecto.

Se han seleccionado dos *datasets* para trabajar con ellos:

- [2] Dataset con anotaciones
- [3] Dataset sin anotaciones

3.1 Preparación y análisis de los datos

El primer *dataset* [2] cuenta con 6024 imágenes, cada una con un archivo JSON con anotaciones.

Veamos un ejemplo:

Imagen 11: Imagen contenida en el dataset [2]

```
{"FileName":"0049.jpg","NumOfAnno":7,"Annotations":
[{"isProtected":false,"ID":833379245415066752,"BoundingBox":
[252,30,324,123],"classname":"face_with_mask","Confidence":1,"Attributes":
{}},{"isProtected":false,"ID":586656586032318720,"BoundingBox":
[249,56,307,79],"classname":"sunglasses","Confidence":1,"Attributes":
{}},{"isProtected":false,"ID":268841533057717408,"BoundingBox":
[257,74,310,122],"classname":"mask_surgical","Confidence":1,"Attributes":
{}},{"isProtected":false,"ID":274990899376364000,"BoundingBox":
[157,59,242,168],"classname":"hijab_niqab","Confidence":1,"Attributes":
{}},{"isProtected":false,"ID":520672409068759232,"BoundingBox":
[157,100,208,122],"classname":"sunglasses","Confidence":1,"Attributes":
{}},{"isProtected":false,"ID":191810024294420416,"BoundingBox":
[159,83,209,157],"classname":"face_other_covering","Confidence":1,"Attributes":
{}},{"isProtected":false,"ID":174243948100625056,"BoundingBox":
[244,19,363,133],"classname":"turban","Confidence":1,"Attributes":{}}]}
```


Imagen 12: Archivo JSON asociado a la imagen

El archivo asociado a esta imagen en concreto consta de las características principales siguientes:

- *Filename*: nombre del archivo
- *NumOfAnno*: número de anotaciones que hay sobre la imagen
- *Annotations*: las diferentes anotaciones, cada anotación hace referencia a un cuadro delimitador (*bounding box*)

  Dentro de la variable *Annotations*, se encuentra lo que nos interesa:
- BoundingBox: coordenadas de donde se encuentra el cuadro delimitador dentro de la imagen
- classname: etiqueta de la clase a la cual pertenece lo detectado dentro del *bounding box*

Estas son las posibles etiquetas de clases que podemos encontrar en los archivos: *hijab_niqab, mask_colorful, mask_surgical, face_no_mask, face_with_mask_incorrect, face_with_mask, face_other_covering, scarf_bandana, balaclava_ski_mask, face_shield, other, gas_mask, turban, helmet, sunglasses, eyeglasses, hair_net, hat, goggles, hood*.

Para nuestro objetivo vamos a agrupar las clases dentro de unas etiquetas:

- *with_mask*: *face_with_mask, gas_mask, face_shield, mask_surgical, mask_colorful*
- *without_mask*: *hijab_niqab, face_other_covering, scarf_bandana, balaclava_ski_mask, other*
- *incorrect_mask*: *face_with_mask_incorrect*

El segundo *dataset* [3] cuenta con 2079 imágenes, divididas en tres carpetas:

- *incorrect_mask*
- *with_mask*
- *without_mask*

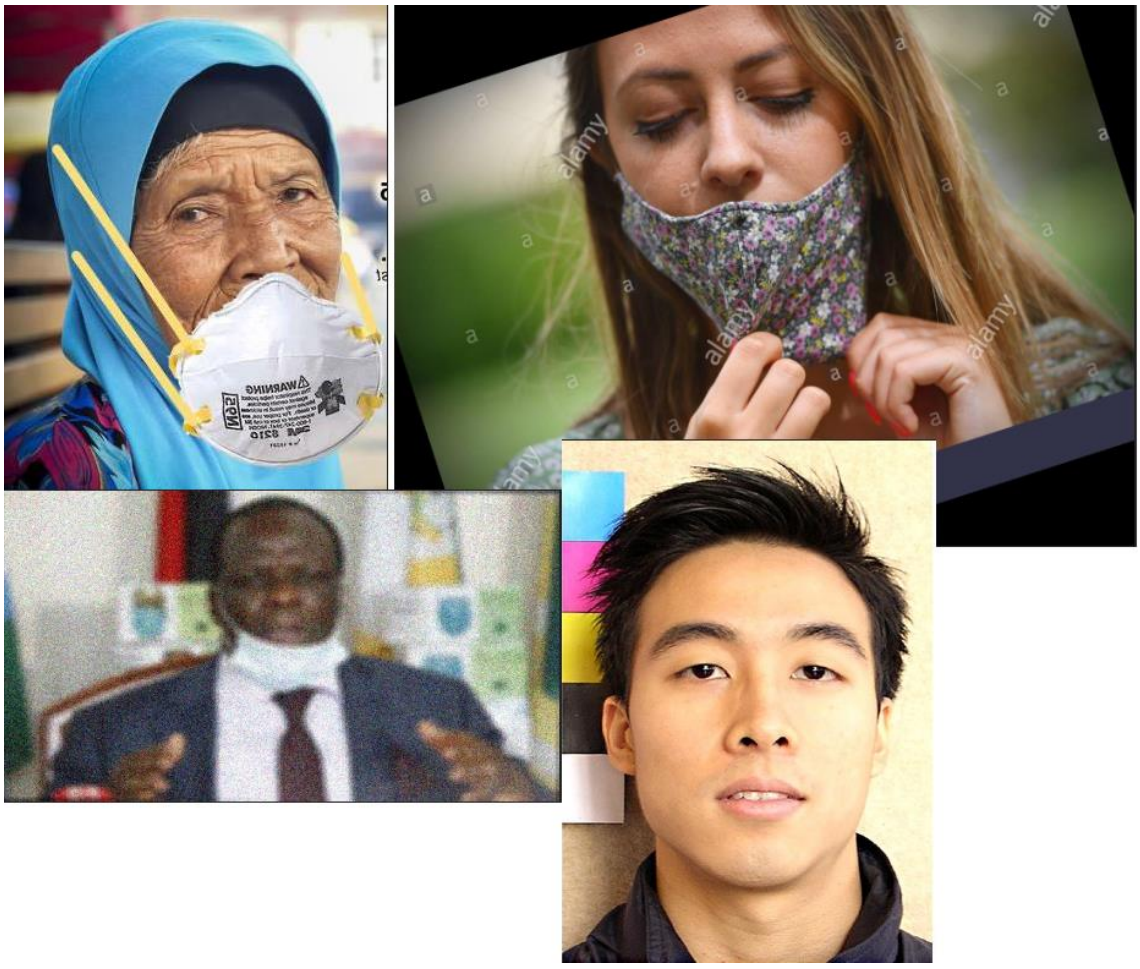


Imagen 13: Pack imágenes del dataset [3]

Este conjunto de datos contiene imágenes con mascarillas colocadas artificialmente e imágenes con *data augmentation* (técnica que permite ampliar un *dataset* creando nuevas imágenes a partir de una origen aplicando diferentes filtros o rotaciones)

3.2 Visualización y distribución de los datos

Se van a analizar y visualizar los datos por separado, es decir, primero analizaremos el dataset [2] y luego el [3]. La decisión de analizarlos por separado viene dada por la estrategia que se ha seguido para entrenar y probar el modelo como explicaremos en el apartado 4.

```
df_labels=df_labels.reset_index()
df_labels['Label'].value_counts()
```

```
with_mask          3912
without_mask       1850
incorrect_mask        6
```

Imagen 14: Distribución de clases dataset [2]

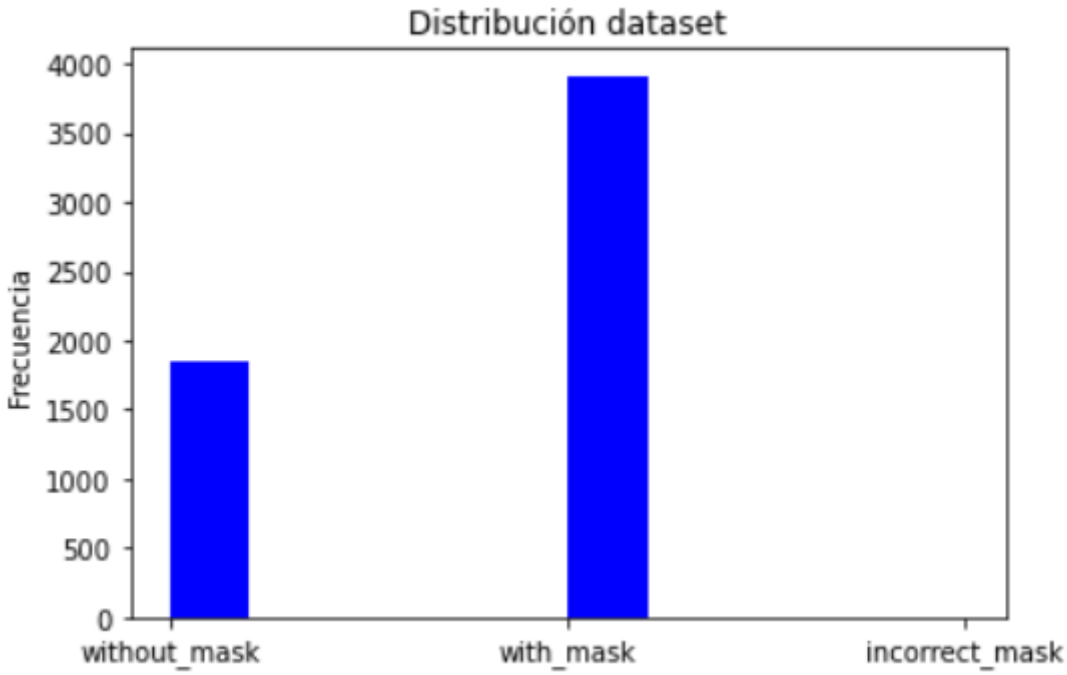


Gráfico 1: Distribución clases dataset [2]

Como se puede observar, las clases están bastante desbalanceadas, concretamente obtenemos 3912 con mascarillas, 1850 sin mascarillas y 6 con la mascarilla incorrectamente colocada.

Si se hacen los cálculos nos salen 5768 imágenes cuando el *dataset* original contaba con 6024, esto se debe a que ha habido imágenes que no se han podido cargar al sistema.

En el siguiente conjunto de datos vemos algo totalmente diferente

```
df_labels['Label'].value_counts()
```

```
incorrect_mask    703
with_mask         690
without_mask      686
```

Imagen 15: Distribución de clases dataset [3]

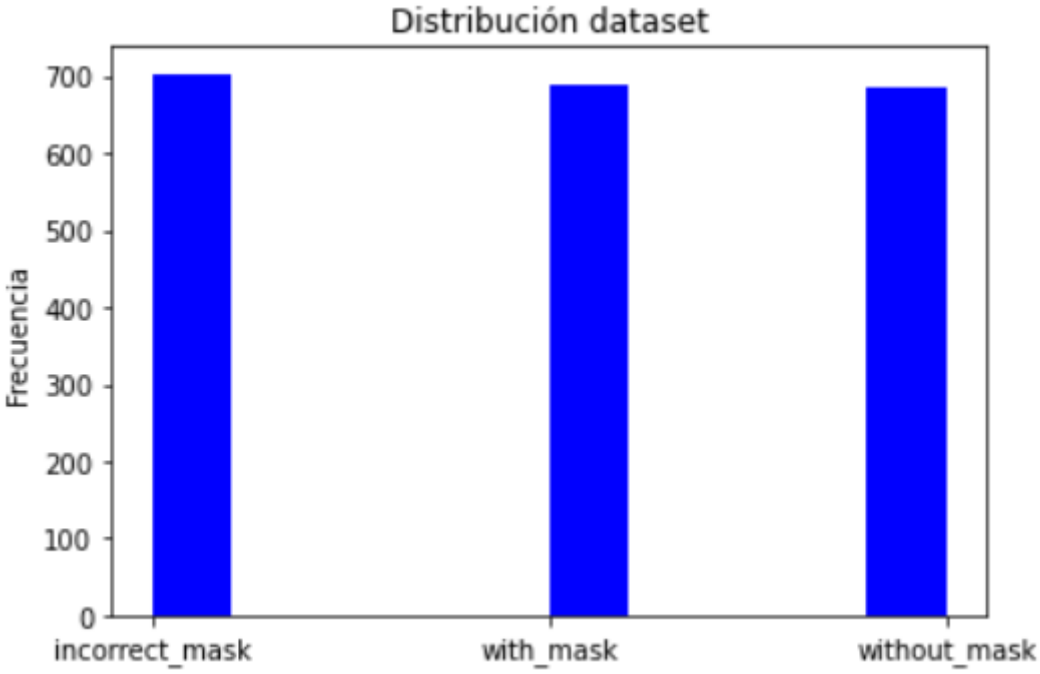


Gráfico 2: Distribución de clases dataset [3]

En este caso obtenemos un dataset bien balanceado pero con pocos registros comparado con el anterior.

3.3 Pre procesado de las imágenes

Antes de empezar a entrenar al modelo, se deben pre procesar las imágenes. Esto se debe a que la mayoría de las redes convolucionales están diseñadas para aceptar imágenes de una medida determinada. Como hemos visto en capítulos anteriores, la entrada de una red neuronal debe ser siempre del mismo tamaño y esto es un hándicap cuando trabajamos con imágenes ya que no todas son de la misma medida.

Una imagen se compone de tres componentes matriciales, uno para cada canal de color RGB (*Red, Green y Blue*). Por ejemplo una imagen de 260x194 pixeles tendrá la forma 260x194x3

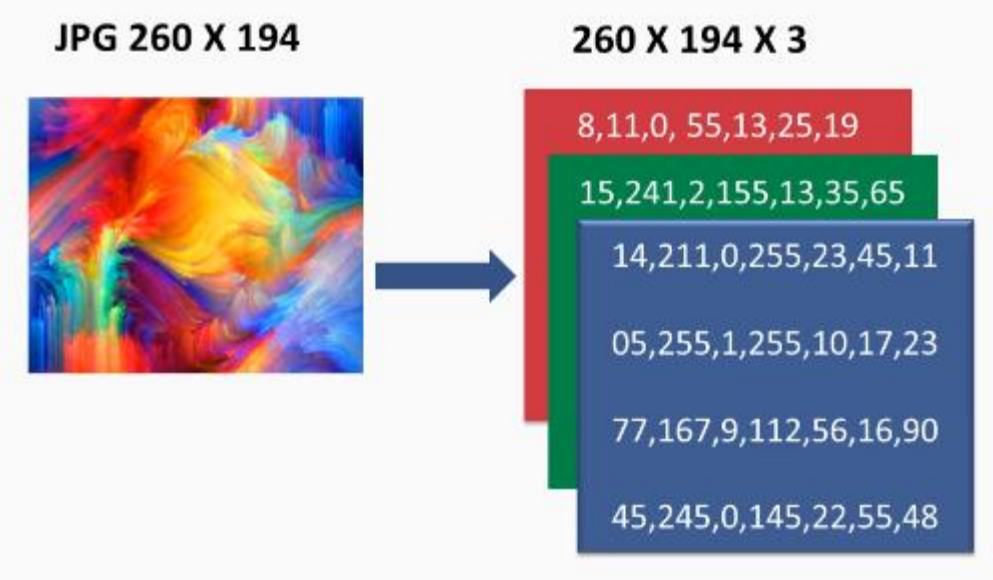


Imagen 16: Imagen con sus canales RGB. Fuente: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781789613964/2/ch02lvl1sec21/convolution-on-rgb-images

En este proyecto se ha reutilizado el modelo *MobileNetV2* que acepta como entrada tres canales desde 32x32x3 hasta 224x224x3. Por lo que se han remodelado todas las imágenes para que todas tengan la forma 224x224x3. Los valores que puede adoptar cada pixel van desde el 0 al 255 así que se normalizan los valores de las matrices para escalarlas entre -1,1.

El siguiente paso será codificar las etiquetas de las imágenes, es decir, cada imagen pertenece a una clase que puede ser: "*incorrect_mask*", "*with_mask*" o "*without_mask*". Para esta codificación vamos a utilizar un método muy simple que nos ofrece *LabelEncoder* (función dentro de la librería de pre procesado de *SciKit learn*), con esta técnica vamos a conseguir valores de 0 y 1 para las clases, una posible codificación sería:

with_mask	0	0	1
without_mask	0	1	0
incorrect_mask	1	0	0

4. Entrenamiento y validación del modelo

Cuando se haya pre procesado todo el conjunto de las imágenes y las etiquetas, estamos en posición para empezar a entrenar el modelo. Lo primero que se hará será dividir el conjunto de datos en dos, una distribución de 80% para el entreno en sí y un 20% para *testing.*

Como se explicó en el apartado 3.1, aplicaremos un aumento de datos sobre el conjunto de entreno (rotaremos, aplicaremos zoom, voltearemos horizontalmente, entre otros filtros para cada imagen)

Para la arquitectura de la CNN, ya se ha expuesto que se utilizará la red *MobileNetV2*. Se va a descartar la última capa de la red, ya que es la encargada de clasificar imágenes sobre el conjunto de datos y etiquetas que fue entrenada.

La arquitectura de nuestra red, que se acoplará a la de *MobileNetV2* es la siguiente:

- Entrada: será la salida de *MobileNetV2* sin contar la última capa que hemos descartado
- Capa de averagePooling de 7x7
- Capa de aplanado o *flattening*
- Capa totalmente conectada de 256 neuronas con función de activación ReLU
- Capa de *dropout* con un 50% de probabilidad
- Capa totalmente conectada de 3 neuronas con función de activación Softmax para hacer la clasificación.

La arquitectura final de la CNN se puede encontrar en los Anexos de este documento.

El siguiente paso es compilar el modelo, vamos a utilizar el optimizador “Adam” con un índice de aprendizaje de 1^{e-4} y una decadencia de 1^{e-3}. Estos parámetros determinan la velocidad a la que nuestra red irá aprendiendo a medida que pasen las iteraciones. Aplicaremos precisión (*accuracy*) como métrica y entropía cruzada categórica (*categorical crossentropy*) como función de pérdidas.

Por último se ajusta el modelo y da comienzo el entreno, vamos a parametrizar que el modelo entrene sobre 10 *epochs* (una *epoch* es una iteración sobre la totalidad de los datos), con un número de muestras calculado como el total de las imágenes de entreno dividido el tamaño del lote (*batch_size*, en nuestro caso 32) por iteración. Al final de cada iteración habrá una fracción de los datos sobre los que se evaluará la pérdida y calculará la métrica (*validation_data*) para propagar el error hacia atrás y recalcular los pesos de las neuronas con el fin de minimizar el error en futuras iteraciones.

Se han probado diferentes configuraciones sobre el modelo:

- se ha entrenado sobre los dos *datasets*
    - Modelo sin fine-tuning (*10 epochs*)
    - Modelo con fine-tuning (*15 epochs*)
- se ha entrenado sobre el dataset [3]
    - Modelo sin fine-tuning (*10 epochs*)
    - Modelo con fine-tuning (*15 epochs*)

Vamos a ver los resultados obtenidos en la fase de entrenamiento con las diferentes configuraciones aplicadas:

Gráfico 3: Pérdida y precisión dataset [3] fine-tuning

Gráfico 4: Pérdida y precisión dataset [3]

Classification Report

	precision	recall	f1-score	support
incorrect_mask	1.00	0.94	0.97	141
with_mask	0.96	1.00	0.98	138
without_mask	0.99	1.00	0.99	137
accuracy			0.98	416
macro avg	0.98	0.98	0.98	416
weighted avg	0.98	0.98	0.98	416

Confusion Matrix

```
[[133   6   2]
 [  0 138   0]
 [  0   0 137]]
```

Tabla 1: Reporte y matriz de confusión dataset [3] fine-tuning

Classification Report

	precision	recall	f1-score	support
incorrect_mask	0.99	0.99	0.99	141
with_mask	0.99	1.00	1.00	138
without_mask	0.99	0.99	0.99	137
accuracy			0.99	416
macro avg	0.99	0.99	0.99	416
weighted avg	0.99	0.99	0.99	416

Confusion Matrix

```
[[139   1   1]
 [  0 138   0]
 [  1   0 136]]
```

Tabla 2: Reporte y matriz de confusión dataset [3]

Analizando la matriz de confusión se puede observar cómo ha sido el *testing* de la red:

Para el *dataset* [3] aplicando *fine-tuning*:

- Datos para test: 416
- incorrect_mask:
  - Bien clasificados: 133
  - Clasificados como with_mask: 6
  - Clasificados como without_mask: 2

- with_mask:
    - Bien clasificados: 138
- without_mask:
    - Bien clasificados: 137

Para el *dataset* [3] sin *fine-tuning*:

- Datos para test: 416
- incorrect_mask:
    - Bien clasificados: 139
    - Clasificados como with_mask: 1
    - Clasificados como without_mask: 1
- with_mask:
    - Bien clasificados: 138
- without_mask:
    - Bien clasificados: 136
    - Clasificados como incorrect_mask: 1

En cuanto al reporte de clasificación podemos centrarnos solamente en el *f1-score* que hace referencia a la precisión del modelo, es la media armónica entre las columnas precisión y *recall*. En el modelo con un ajuste fino obtenemos un 98% de media, mientras que en la red sin ajuste un 99%. Probablemente se deba a que el conjunto de entreno no es lo suficientemente extenso como para generalizar los pesos de las capas que se han descongelado en el proceso.

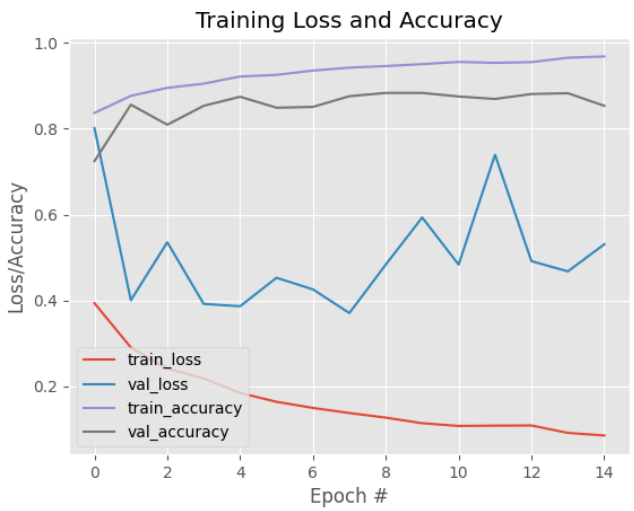


Gráfico 5: Pérdida y precisión dataset [2] fine-tuning

Gráfico 6: Pérdida y precisión dataset [2] fine-tuning

Classification Report

	precision	recall	f1-score	support
incorrect_mask	0.93	0.97	0.95	142
with_mask	0.91	0.84	0.87	921
without_mask	0.76	0.85	0.80	507
accuracy			0.86	1570
macro avg	0.86	0.89	0.87	1570
weighted avg	0.86	0.86	0.86	1570

Confusion Matrix

```
[[138   4   0]
 [  9 775 137]
 [  2  75 430]]
```

Tabla 3: Reporte y matriz de confusión dataset [2] fine-tuning

Classification Report

	precision	recall	f1-score	support
incorrect_mask	0.93	0.96	0.95	142
with_mask	0.87	0.92	0.89	921
without_mask	0.85	0.75	0.80	507
accuracy			0.87	1570
macro avg	0.88	0.88	0.88	1570
weighted avg	0.87	0.87	0.87	1570

Confusion Matrix

```
[[137   5   0]
 [  7 845  69]
 [  3 123 381]]
```

Tabla 4: Reporte y matriz de confusión dataset [2]

Para el dataset [2] aplicando fine-tuning:

- Datos para test: 1570
- incorrect_mask:
  - Bien clasificados: 138
  - Clasificados como with_mask: 4
- with_mask:
  - Bien clasificados: 775
  - Clasificados como incorrect_mask: 9
  - Clasificados como without_mask: 137
- without_mask:
  - Bien clasificados: 430

    - Clasificados como incorrect_mask: 2
    - Clasificados como with_mask: 75

Para el dataset [2] sin fine-tuning:

- incorrect_mask:
    - Bien clasificados: 137
    - Clasificados como with_mask: 5
- with_mask:
    - Bien clasificados: 845
    - Clasificados como incorrect_mask: 7
    - Clasificados como without_mask: 69
- without_mask:
    - Bien clasificados: 381
    - Clasificados como incorrect_mask: 3
    - Clasificados como with_mask: 123

En cuanto al reporte de clasificación en el modelo con un ajuste fino se obtiene un 86% de media, mientras que en la red sin ajuste un 87%.

5. Pruebas con vídeo en tiempo real

La demo con las pruebas del modelo con vídeo en tiempo real se encuentran alojadas en Google Drive.

Se puede acceder mediante el siguiente enlace (se recomienda reproducir los videos con sonido):

https://drive.google.com/drive/folders/1zIA7U6ukgL5GehRA4Vz3CNsVa4Mc37Ya?usp=sharing

Para mi sorpresa el modelo que mejor se comporta es el que ha sido entrenado sobre el conjunto de datos [3] y sin haberle aplicado un ajuste fino a la red.

6. Conclusiones

Durante la realización del proyecto he adquirido mucho conocimiento de un campo muy interesante como es el de la visión por computador. Las asignaturas que había cursado sobre inteligencia artificial ninguna de ellas profundizaba o explicaba técnicas de *Deep learning,* muchas estaban centradas en la parte más estadística del aprendizaje automático o la minería de datos.

Adentrarme en esta rama de la IA me ha permitido conocer cómo funcionan las CNN, cómo se puede reaprovechar un conocimiento adquirido previamente para adaptarlo a nuestras necesidades y el potencial que tiene esta tecnología en el día a día ya sea en el ámbito sanitario, industrial o logístico entre otros.

Los resultados obtenidos en este proyecto se pueden mejorar no hay duda, pero creo que se ha conseguido un muy buen rendimiento. El objetivo principal por el que fue desarrollado este proyecto lo ha cumplido satisfactoriamente, el cual es detectar si una persona lleva una mascarilla, si no la lleva o si la lleva mal colocada.

En cuanto a la planificación y metodología se ha seguido sin muchos problemas ni desviaciones. Algún pequeño retraso a consecuencia de algún bug o por añadir funciones que no estaban previstas en un momento, pero nada alarmante como para influir en las fechas estimadas.

A vista futura el proyecto puede quedar en un limbo, cuando las mascarillas no sean obligatorias el propósito de esta aplicación podría quedar obsoleta. No obstante, se puede reaprovechar todo el ecosistema para adaptarlo a otras tareas por ejemplo, que en una obra todas las personas lleven las medidas de seguridad (casco, chaleco reflectante, etc.)

7. Glosario

1. Adam: Derivado de Adaptative Moment Estimation, es un algoritmo de optimización que se basa en el descenso del gradiente.
2. Capa: Referente a las redes neuronales, es un conjunto de neuronas cuyas entradas provienen de una capa anterior y sus salidas son las entradas de una capa posterior
3. Dataset: Se trata de un conjunto de datos
4. Deep learning: es una rama del aprendizaje automático, gracias al cual los ordenadores pueden procesar grandes cantidades de datos con la ayuda de redes neuronales inspiradas en el cerebro humano con el fin de adquirir ciertos conocimientos
5. Input: entrada
6. Json: acrónimo de JavaScript Object Notation, es un formato de texto sencillo para el intercambio de datos
7. Keras: proporciona un acceso de alto nivel mediante una API a las funcionalidades de la librería TensorFlow, CNTK, o Theano.
8. Matriz de confusión: es una herramienta que permite la visualización del desempeño de un algoritmo que se emplea en aprendizaje supervisado
9. Scikit-learn: es una biblioteca para aprendizaje automático de software libre para el lenguaje de programación Python.

8. Bibliografía

[1] https://github.com/balajisrinivas/Face-Mask-Detection

[2] https://blog.infaimon.com/vision-computador-soluciones-permite/

[3] https://informatica.blogs.uoc.edu/la-vision-por-computador-una-disciplina-en-auge/

[4] https://ciberninjas.com/vision-computadora-agosto-2020/

[5] https://ciberninjas.com/vision-computadora-agosto-2020/#aplicaciones-de-visi%C3%B3n-artificial

[6] https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/

[7] https://www.diegocalvo.es/red-neuronal-convolucional/

[8] https://relopezbriega.github.io/blog/2016/08/02/redes-neuronales-convolucionales-con-tensorflow/

[9] https://www.researchgate.net/figure/Illustration-of-Max-Pooling-and-Average-Pooling-Figure-2-above-shows-an-example-of-max_fig2_333593451

[10] https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening

[11] https://empresas.blogthinkbig.com/transfer-learning-en-modelos-profundos/

[12] https://www.tensorflow.org/tutorials/images/transfer_learning

[13] https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

[14] https://keras.rstudio.com/reference/application_mobilenet_v2.html

[15] https://keras.io/api/models/model_training_apis/

[16] https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial

9. Anexos

Repositorio del código:

https://github.com/EzequielCuadra/TFG

Arquitectura de la CNN final:

Model: "model"

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
Conv1 (Conv2D)                  (None, 112, 112, 32) 864         input_1[0][0]
__________________________________________________________________________________________________
bn_Conv1 (BatchNormalization)   (None, 112, 112, 32) 128         Conv1[0][0]
__________________________________________________________________________________________________
Conv1_relu (ReLU)               (None, 112, 112, 32) 0           bn_Conv1[0][0]
__________________________________________________________________________________________________
expanded_conv_depthwise (Depthw (None, 112, 112, 32) 288         Conv1_relu[0][0]
__________________________________________________________________________________________________
expanded_conv_depthwise_BN (Bat (None, 112, 112, 32) 128         expanded_conv_depthwise[0][0]
__________________________________________________________________________________________________
expanded_conv_depthwise_relu (R (None, 112, 112, 32) 0           expanded_conv_depthwise_BN[0][0]
__________________________________________________________________________________________________
expanded_conv_project (Conv2D)  (None, 112, 112, 16) 512         expanded_conv_depthwise_relu[0][0
__________________________________________________________________________________________________
expanded_conv_project_BN (Batch (None, 112, 112, 16) 64          expanded_conv_project[0][0]
__________________________________________________________________________________________________
block_1_expand (Conv2D)         (None, 112, 112, 96) 1536        expanded_conv_project_BN[0][0]
__________________________________________________________________________________________________
block_1_expand_BN (BatchNormali (None, 112, 112, 96) 384         block_1_expand[0][0]
__________________________________________________________________________________________________
block_1_expand_relu (ReLU)      (None, 112, 112, 96) 0           block_1_expand_BN[0][0]
__________________________________________________________________________________________________
block_1_pad (ZeroPadding2D)     (None, 113, 113, 96) 0           block_1_expand_relu[0][0]
__________________________________________________________________________________________________
block_1_depthwise (DepthwiseCon (None, 56, 56, 96)   864         block_1_pad[0][0]
__________________________________________________________________________________________________
block_1_depthwise_BN (BatchNorm (None, 56, 56, 96)   384         block_1_depthwise[0][0]
__________________________________________________________________________________________________
block_1_depthwise_relu (ReLU)   (None, 56, 56, 96)   0           block_1_depthwise_BN[0][0]
__________________________________________________________________________________________________
block_1_project (Conv2D)        (None, 56, 56, 24)   2304        block_1_depthwise_relu[0][0]
__________________________________________________________________________________________________
block_1_project_BN (BatchNormal (None, 56, 56, 24)   96          block_1_project[0][0]
__________________________________________________________________________________________________
block_2_expand (Conv2D)         (None, 56, 56, 144)  3456        block_1_project_BN[0][0]
__________________________________________________________________________________________________
block_2_expand_BN (BatchNormali (None, 56, 56, 144)  576         block_2_expand[0][0]
__________________________________________________________________________________________________
block_2_expand_relu (ReLU)      (None, 56, 56, 144)  0           block_2_expand_BN[0][0]
__________________________________________________________________________________________________
block_2_depthwise (DepthwiseCon (None, 56, 56, 144)  1296        block_2_expand_relu[0][0]
__________________________________________________________________________________________________
block_2_depthwise_BN (BatchNorm (None, 56, 56, 144)  576         block_2_depthwise[0][0]
__________________________________________________________________________________________________
block_2_depthwise_relu (ReLU)   (None, 56, 56, 144)  0           block_2_depthwise_BN[0][0]
__________________________________________________________________________________________________
block_2_project (Conv2D)        (None, 56, 56, 24)   3456        block_2_depthwise_relu[0][0]
__________________________________________________________________________________________________
block_2_project_BN (BatchNormal (None, 56, 56, 24)   96          block_2_project[0][0]
```

block_2_add (Add)	(None, 56, 56, 24)	0	block_1_project_BN[0][0] block_2_project_BN[0][0]
block_3_expand (Conv2D)	(None, 56, 56, 144)	3456	block_2_add[0][0]
block_3_expand_BN (BatchNormali	(None, 56, 56, 144)	576	block_3_expand[0][0]
block_3_expand_relu (ReLU)	(None, 56, 56, 144)	0	block_3_expand_BN[0][0]
block_3_pad (ZeroPadding2D)	(None, 57, 57, 144)	0	block_3_expand_relu[0][0]
block_3_depthwise (DepthwiseCon	(None, 28, 28, 144)	1296	block_3_pad[0][0]
block_3_depthwise_BN (BatchNorm	(None, 28, 28, 144)	576	block_3_depthwise[0][0]
block_3_depthwise_relu (ReLU)	(None, 28, 28, 144)	0	block_3_depthwise_BN[0][0]
block_3_project (Conv2D)	(None, 28, 28, 32)	4608	block_3_depthwise_relu[0][0]
block_3_project_BN (BatchNormal	(None, 28, 28, 32)	128	block_3_project[0][0]
block_4_expand (Conv2D)	(None, 28, 28, 192)	6144	block_3_project_BN[0][0]
block_4_expand_BN (BatchNormali	(None, 28, 28, 192)	768	block_4_expand[0][0]
block_4_expand_relu (ReLU)	(None, 28, 28, 192)	0	block_4_expand_BN[0][0]
block_4_depthwise (DepthwiseCon	(None, 28, 28, 192)	1728	block_4_expand_relu[0][0]
block_4_depthwise_BN (BatchNorm	(None, 28, 28, 192)	768	block_4_depthwise[0][0]
block_4_depthwise_relu (ReLU)	(None, 28, 28, 192)	0	block_4_depthwise_BN[0][0]
block_4_project (Conv2D)	(None, 28, 28, 32)	6144	block_4_depthwise_relu[0][0]
block_4_project_BN (BatchNormal	(None, 28, 28, 32)	128	block_4_project[0][0]
block_4_add (Add)	(None, 28, 28, 32)	0	block_3_project_BN[0][0] block_4_project_BN[0][0]
block_5_expand (Conv2D)	(None, 28, 28, 192)	6144	block_4_add[0][0]
block_5_expand_BN (BatchNormali	(None, 28, 28, 192)	768	block_5_expand[0][0]
block_5_expand_relu (ReLU)	(None, 28, 28, 192)	0	block_5_expand_BN[0][0]
block_5_depthwise (DepthwiseCon	(None, 28, 28, 192)	1728	block_5_expand_relu[0][0]
block_5_depthwise_BN (BatchNorm	(None, 28, 28, 192)	768	block_5_depthwise[0][0]
block_5_depthwise_relu (ReLU)	(None, 28, 28, 192)	0	block_5_depthwise_BN[0][0]
block_5_project (Conv2D)	(None, 28, 28, 32)	6144	block_5_depthwise_relu[0][0]
block_5_project_BN (BatchNormal	(None, 28, 28, 32)	128	block_5_project[0][0]
block_5_add (Add)	(None, 28, 28, 32)	0	block_4_add[0][0] block_5_project_BN[0][0]
block_6_expand (Conv2D)	(None, 28, 28, 192)	6144	block_5_add[0][0]
block_6_expand_BN (BatchNormali	(None, 28, 28, 192)	768	block_6_expand[0][0]
block_6_expand_relu (ReLU)	(None, 28, 28, 192)	0	block_6_expand_BN[0][0]
block_6_pad (ZeroPadding2D)	(None, 29, 29, 192)	0	block_6_expand_relu[0][0]
block_6_depthwise (DepthwiseCon	(None, 14, 14, 192)	1728	block_6_pad[0][0]
block_6_depthwise_BN (BatchNorm	(None, 14, 14, 192)	768	block_6_depthwise[0][0]

block_6_depthwise_relu (ReLU) (None, 14, 14, 192) 0 block_6_depthwise_BN[0][0]

block_6_project (Conv2D) (None, 14, 14, 64) 12288 block_6_depthwise_relu[0][0]

block_6_project_BN (BatchNormal (None, 14, 14, 64) 256 block_6_project[0][0]

block_7_expand (Conv2D) (None, 14, 14, 384) 24576 block_6_project_BN[0][0]

block_7_expand_BN (BatchNormali (None, 14, 14, 384) 1536 block_7_expand[0][0]

block_7_expand_relu (ReLU) (None, 14, 14, 384) 0 block_7_expand_BN[0][0]

block_7_depthwise (DepthwiseCon (None, 14, 14, 384) 3456 block_7_expand_relu[0][0]

block_7_depthwise_BN (BatchNorm (None, 14, 14, 384) 1536 block_7_depthwise[0][0]

block_7_depthwise_relu (ReLU) (None, 14, 14, 384) 0 block_7_depthwise_BN[0][0]

block_7_project (Conv2D) (None, 14, 14, 64) 24576 block_7_depthwise_relu[0][0]

block_7_project_BN (BatchNormal (None, 14, 14, 64) 256 block_7_project[0][0]

block_7_add (Add) (None, 14, 14, 64) 0 block_6_project_BN[0][0]
block_7_project_BN[0][0]

block_8_expand (Conv2D) (None, 14, 14, 384) 24576 block_7_add[0][0]

block_8_expand_BN (BatchNormali (None, 14, 14, 384) 1536 block_8_expand[0][0]

block_8_expand_relu (ReLU) (None, 14, 14, 384) 0 block_8_expand_BN[0][0]

block_8_depthwise (DepthwiseCon (None, 14, 14, 384) 3456 block_8_expand_relu[0][0]

block_8_depthwise_BN (BatchNorm (None, 14, 14, 384) 1536 block_8_depthwise[0][0]

block_8_depthwise_relu (ReLU) (None, 14, 14, 384) 0 block_8_depthwise_BN[0][0]

block_8_project (Conv2D) (None, 14, 14, 64) 24576 block_8_depthwise_relu[0][0]

block_8_project_BN (BatchNormal (None, 14, 14, 64) 256 block_8_project[0][0]

block_8_add (Add) (None, 14, 14, 64) 0 block_7_add[0][0]
block_8_project_BN[0][0]

block_9_expand (Conv2D) (None, 14, 14, 384) 24576 block_8_add[0][0]

block_9_expand_BN (BatchNormali (None, 14, 14, 384) 1536 block_9_expand[0][0]

block_9_expand_relu (ReLU) (None, 14, 14, 384) 0 block_9_expand_BN[0][0]

block_9_depthwise (DepthwiseCon (None, 14, 14, 384) 3456 block_9_expand_relu[0][0]

block_9_depthwise_BN (BatchNorm (None, 14, 14, 384) 1536 block_9_depthwise[0][0]

block_9_depthwise_relu (ReLU) (None, 14, 14, 384) 0 block_9_depthwise_BN[0][0]

block_9_project (Conv2D) (None, 14, 14, 64) 24576 block_9_depthwise_relu[0][0]

block_9_project_BN (BatchNormal (None, 14, 14, 64) 256 block_9_project[0][0]

block_9_add (Add) (None, 14, 14, 64) 0 block_8_add[0][0]
block_9_project_BN[0][0]

block_10_expand (Conv2D) (None, 14, 14, 384) 24576 block_9_add[0][0]

block_10_expand_BN (BatchNormal (None, 14, 14, 384) 1536 block_10_expand[0][0]

block_10_expand_relu (ReLU) (None, 14, 14, 384) 0 block_10_expand_BN[0][0]

block_10_depthwise (DepthwiseCo (None, 14, 14, 384) 3456 block_10_expand_relu[0][0]

block_10_depthwise_BN (BatchNor (None, 14, 14, 384) 1536 block_10_depthwise[0][0]

block_10_depthwise_relu (ReLU)	(None, 14, 14, 384)	0	block_10_depthwise_BN[0][0]
block_10_project (Conv2D)	(None, 14, 14, 96)	36864	block_10_depthwise_relu[0][0]
block_10_project_BN (BatchNorma	(None, 14, 14, 96)	384	block_10_project[0][0]
block_11_expand (Conv2D)	(None, 14, 14, 576)	55296	block_10_project_BN[0][0]
block_11_expand_BN (BatchNormal	(None, 14, 14, 576)	2304	block_11_expand[0][0]
block_11_expand_relu (ReLU)	(None, 14, 14, 576)	0	block_11_expand_BN[0][0]
block_11_depthwise (DepthwiseCo	(None, 14, 14, 576)	5184	block_11_expand_relu[0][0]
block_11_depthwise_BN (BatchNor	(None, 14, 14, 576)	2304	block_11_depthwise[0][0]
block_11_depthwise_relu (ReLU)	(None, 14, 14, 576)	0	block_11_depthwise_BN[0][0]
block_11_project (Conv2D)	(None, 14, 14, 96)	55296	block_11_depthwise_relu[0][0]
block_11_project_BN (BatchNorma	(None, 14, 14, 96)	384	block_11_project[0][0]
block_11_add (Add)	(None, 14, 14, 96)	0	block_10_project_BN[0][0] block_11_project_BN[0][0]
block_12_expand (Conv2D)	(None, 14, 14, 576)	55296	block_11_add[0][0]
block_12_expand_BN (BatchNormal	(None, 14, 14, 576)	2304	block_12_expand[0][0]
block_12_expand_relu (ReLU)	(None, 14, 14, 576)	0	block_12_expand_BN[0][0]
block_12_depthwise (DepthwiseCo	(None, 14, 14, 576)	5184	block_12_expand_relu[0][0]
block_12_depthwise_BN (BatchNor	(None, 14, 14, 576)	2304	block_12_depthwise[0][0]
block_12_depthwise_relu (ReLU)	(None, 14, 14, 576)	0	block_12_depthwise_BN[0][0]
block_12_project (Conv2D)	(None, 14, 14, 96)	55296	block_12_depthwise_relu[0][0]
block_12_project_BN (BatchNorma	(None, 14, 14, 96)	384	block_12_project[0][0]
block_12_add (Add)	(None, 14, 14, 96)	0	block_11_add[0][0] block_12_project_BN[0][0]
block_13_expand (Conv2D)	(None, 14, 14, 576)	55296	block_12_add[0][0]
block_13_expand_BN (BatchNormal	(None, 14, 14, 576)	2304	block_13_expand[0][0]
block_13_expand_relu (ReLU)	(None, 14, 14, 576)	0	block_13_expand_BN[0][0]
block_13_pad (ZeroPadding2D)	(None, 15, 15, 576)	0	block_13_expand_relu[0][0]
block_13_depthwise (DepthwiseCo	(None, 7, 7, 576)	5184	block_13_pad[0][0]
block_13_depthwise_BN (BatchNor	(None, 7, 7, 576)	2304	block_13_depthwise[0][0]
block_13_depthwise_relu (ReLU)	(None, 7, 7, 576)	0	block_13_depthwise_BN[0][0]
block_13_project (Conv2D)	(None, 7, 7, 160)	92160	block_13_depthwise_relu[0][0]
block_13_project_BN (BatchNorma	(None, 7, 7, 160)	640	block_13_project[0][0]
block_14_expand (Conv2D)	(None, 7, 7, 960)	153600	block_13_project_BN[0][0]
block_14_expand_BN (BatchNormal	(None, 7, 7, 960)	3840	block_14_expand[0][0]
block_14_expand_relu (ReLU)	(None, 7, 7, 960)	0	block_14_expand_BN[0][0]
block_14_depthwise (DepthwiseCo	(None, 7, 7, 960)	8640	block_14_expand_relu[0][0]
block_14_depthwise_BN (BatchNor	(None, 7, 7, 960)	3840	block_14_depthwise[0][0]

```
block_14_depthwise_relu (ReLU)  (None, 7, 7, 960)    0          block_14_depthwise_BN[0][0]
block_14_project (Conv2D)       (None, 7, 7, 160)    153600     block_14_depthwise_relu[0][0]
block_14_project_BN (BatchNorma (None, 7, 7, 160)    640        block_14_project[0][0]
block_14_add (Add)              (None, 7, 7, 160)    0          block_13_project_BN[0][0]
                                                                block_14_project_BN[0][0]
block_15_expand (Conv2D)        (None, 7, 7, 960)    153600     block_14_add[0][0]
block_15_expand_BN (BatchNormal (None, 7, 7, 960)    3840       block_15_expand[0][0]
block_15_expand_relu (ReLU)     (None, 7, 7, 960)    0          block_15_expand_BN[0][0]
block_15_depthwise (DepthwiseCo (None, 7, 7, 960)    8640       block_15_expand_relu[0][0]
block_15_depthwise_BN (BatchNor (None, 7, 7, 960)    3840       block_15_depthwise[0][0]
block_15_depthwise_relu (ReLU)  (None, 7, 7, 960)    0          block_15_depthwise_BN[0][0]
block_15_project (Conv2D)       (None, 7, 7, 160)    153600     block_15_depthwise_relu[0][0]
block_15_project_BN (BatchNorma (None, 7, 7, 160)    640        block_15_project[0][0]
block_15_add (Add)              (None, 7, 7, 160)    0          block_14_add[0][0]
                                                                block_15_project_BN[0][0]
block_16_expand (Conv2D)        (None, 7, 7, 960)    153600     block_15_add[0][0]
block_16_expand_BN (BatchNormal (None, 7, 7, 960)    3840       block_16_expand[0][0]
block_16_expand_relu (ReLU)     (None, 7, 7, 960)    0          block_16_expand_BN[0][0]
block_16_depthwise (DepthwiseCo (None, 7, 7, 960)    8640       block_16_expand_relu[0][0]
block_16_depthwise_BN (BatchNor (None, 7, 7, 960)    3840       block_16_depthwise[0][0]
block_16_depthwise_relu (ReLU)  (None, 7, 7, 960)    0          block_16_depthwise_BN[0][0]
block_16_project (Conv2D)       (None, 7, 7, 320)    307200     block_16_depthwise_relu[0][0]
block_16_project_BN (BatchNorma (None, 7, 7, 320)    1280       block_16_project[0][0]
Conv_1 (Conv2D)                 (None, 7, 7, 1280)   409600     block_16_project_BN[0][0]
Conv_1_bn (BatchNormalization)  (None, 7, 7, 1280)   5120       Conv_1[0][0]
out_relu (ReLU)                 (None, 7, 7, 1280)   0          Conv_1_bn[0][0]
average_pooling2d (AveragePooli (None, 1, 1, 1280)   0          out_relu[0][0]
flatten (Flatten)               (None, 1280)         0          average_pooling2d[0][0]
dense (Dense)                   (None, 256)          327936     flatten[0][0]
dropout (Dropout)               (None, 256)          0          dense[0][0]
dense_1 (Dense)                 (None, 3)            771        dropout[0][0]
==================================================================================================
Total params: 2,586,691
Trainable params: 328,707
Non-trainable params: 2,257,984
__________________________________________________________________________________________________
```