1. CATS & DOGS

1.1. [1 pts] TODO 1. Añadir una capa de convolución según las especificaciones del código

Respuesta:

```
# TODO 1: Crear una segunda capa convolucional seguida de una MaxPooling2D
# Recuerda que esta convolución debería extraer 32 filtros de 3x3
# Utilizaremos a continuación una capa MaxPooling2D con tamaño de ventana 2x2
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
```

1.2. [1 pts] TODO 2. Vectorizar (flatten) la salida de la última capa convolucional para poder añadir una capa Dense

Respuesta:

```
# TODO 2: Aplana "Flatten" la salida de la capa precedente
# Aplana el feature map a un tensor de 1 dimensión para añadir nuestras capas fully connected
x = layers.Flatten()(x)

# TODO 3: Crea una capa fully-connected (dense)
# Crea una capa fully-connected con función de activación ReLU y 512 neuronas
x = layers.Dense(512, activation='relu')(x)
```

1.3. [1 pts] TODO 3. Añadir nuestra capa DNN o feed-forward (antes de la salida) según las especificaciones del código

Respuesta:

```
# TODO 3: Crea una capa fully-connected (dense)
# Crea una capa fully-connected con función de activación ReLU y 512 neuronas
x = layers.Dense(512, activation='relu')(x)

# Creamos la capa de salida con un sólo nodo y función de activación sigmoide
output = layers.Dense(1, activation='sigmoid')(x)
```

2. Regularización

2.1. [1 pts] TODO 1. Data Augmentation. Explica cómo funciona ImageDataGenerator. Explica específicamente lo que significan cada uno de los parámetros y sus valores. Uno a uno, desde rotation_range hasta fill_mode. La documentación de Keras será tu aliada.

Respuesta:

- La clase **ImageDataGenerator** es una de las clases más importante que tiene Keras para hacer Data Augmentation con imágenes y por lo tanto muy usadas con junto a las redes CNN.
- Se sitúa en la siguiente ruta de la librería ver la **tf.keras.preprocessing.image**
- Lo que realiza, a nivel de resumen, son lotes de imágenes modificadas de las originales, por lo tanto, distintas, gracias a los diversos parámetros con los que se configura. Así, para cada epoch del entrenamiento de la CNN, se entrena con imágenes distintas de la original. Estas modificaciones se realizan ejecutando varias acciones sobre las imágenes (desplazamientos, recortes, rotaciones, invertirlas,), con lo que el dataset de imágenes original se incremente bastante.
- Los parámetros más importantes de **ImageDataGenerator**:
  - **rotation_range:** Int**.** Indica los grados que queremos rotar la imagen. Él valor puede estar en el rango de (0–180).
  - **width_shift_range** y **height_shift_range** : Float/Int. Lo utilizaremos para configurar el rango de desplazamiento tanto horizontal como vertical que queremos que tengan las imágenes. Dependiendo si son float o int estaremos indicando:
    - **Float:** Si son < 1. indica una fracción del width actual. Si es >=1. Indica el número de pixeles a desplazar.
    - **Int:** numero de pixeles del intervalo (-**width_shift_range** , **width_shift_range**) o en el caso del height (-**heigth_shift_range** , **heigth_shift_range**)
  - **shear_range:** Float. Indica los grados del ángulo para aplicar proyecciones de la imagen y así poder tener una imagen con distintos ángulos de visión.
  - **zoom_range:** Float. Es el rango de valores para los que se haría un zoom aleatorio [menor,mayor]. Si solo le informamos un número toma los valores [1-valor,1+valor].
  - **horizontal_flip:** Boolean. Permite voltear de forma aleatoria horizontalmente.
  - **fill_mode**: La forma de relleno de los nuevos pixeles que se puedan generar al realizar transformaciones sobre las imágenes. Las formas de relleno, siendo abcd los pixels originales, pueden ser:
    - **'constant'**: kkkkkkkk|abcd|kkkkkkkk (cval=k)
    - **'nearest'**: aaaaaaaa|abcd|dddddddd
    - **'reflect'**: abcddcba|abcd|dcbaabcd
    - **'wrap'**: abcdabcd|abcd|abcdabcd

2.2. [1,5 pts] TODO 2. Dropout.

Explica Dropout como técnica de regularización.

Respuesta:

- A nivel general la regularización es una técnica que dificulta el aprendizaje de los modelos, evitando así el sobre ajuste.
- El dropout junto a las penalizaciones L1 (Laplaciano) y L2 (Gausiano) es de las técnicas más conocidas para la regularización.
- En concreto el dropout lo que hace, en la fase entrenamiento, y de forma aleatoria con una probabilidad previamente fijada, es que en cada época "apaga" (se genera la entrada igual a 0) un porcentaje de neuronas y sus conexiones, de las distintas capas que contienen la red neuronal. Así se evita que las neuronas memoricen la entrada, ya que les puede cambiar de una época a otra.
- Es recomendable que la tasa que se fija en el dropout, cuyo valor va desde 0 a 1 y que identifica el porcentaje de neuronas a apagar en cada capa, no sea muy alto, se recomienda entre 0,2 y 0,5. Esto es debido a que con valores muy bajos seguiría habiendo sobreajuste y que con valores muy altos la red no aprenderá, es decir produce subajuste.
- El efecto del dropout es que se necesiten más iteraciones, pero se converge más rápidamente
- Una imagen gráfica sería la siguiente:

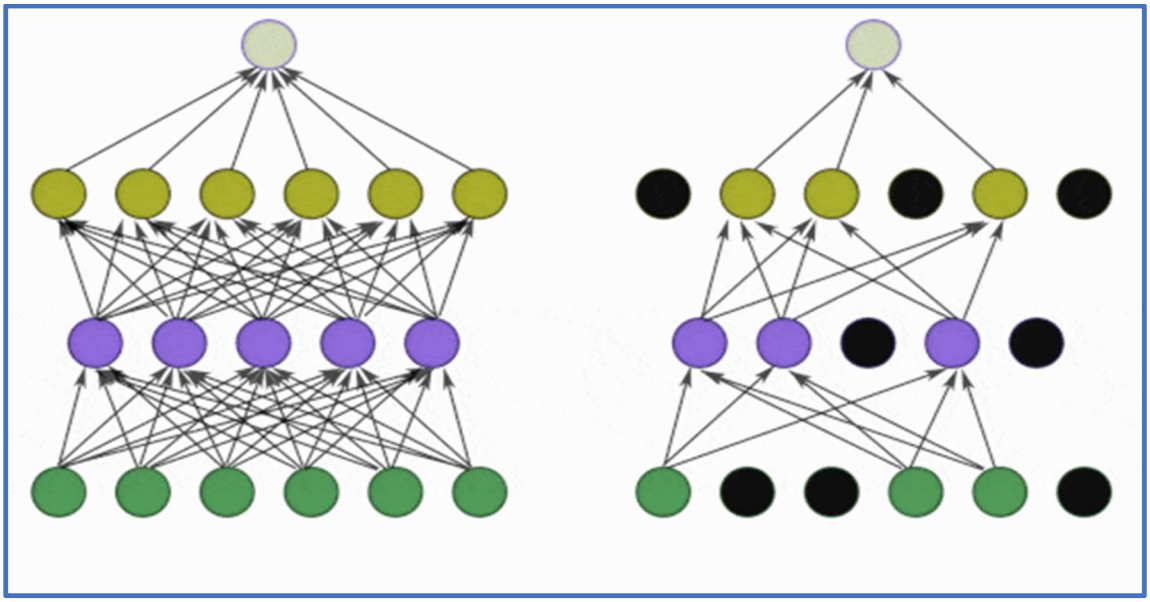
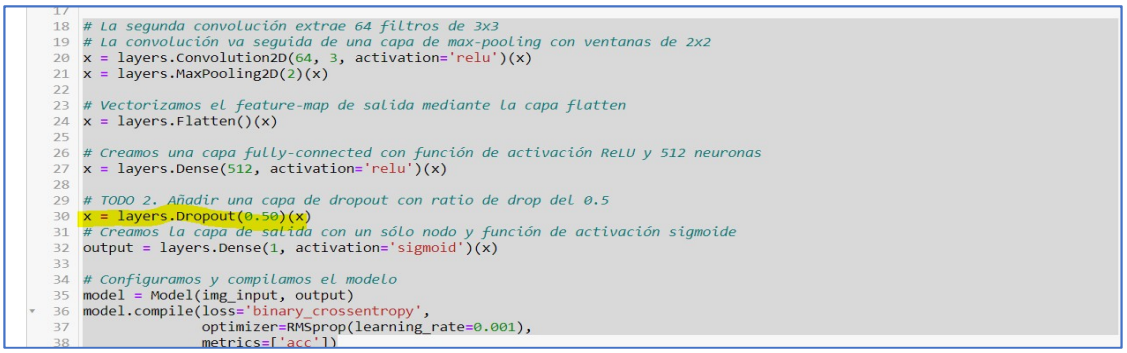


Figura1: Imagen de un esquema de red neuronal en el que a la izquierda se ve un efecto del dropout y de cómo hay neuronas que dejan de estar conectadas.

Añade una capa de dropout con una ratio de 0.5

Respuesta:

Seguidamente se muestra un fragmento del código donde se establece una capa dropout con un arte de 0,5.



```
# La segunda convolución extrae 64 filtros de 3x3
# La convolución va seguida de una capa de max-pooling con ventanas de 2x2
x = layers.Convolution2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)

# Vectorizamos el feature-map de salida mediante la capa flatten
x = layers.Flatten()(x)

# Creamos una capa fully-connected con función de activación ReLU y 512 neuronas
x = layers.Dense(512, activation='relu')(x)

# TODO 2. Añadir una capa de dropout con ratio de drop del 0.5
x = layers.Dropout(0.50)(x)
# Creamos la capa de salida con un sólo nodo y función de activación sigmoide
output = layers.Dense(1, activation='sigmoid')(x)

# Configuramos y compilamos el modelo
model = Model(img_input, output)
model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(learning_rate=0.001),
              metrics=['acc'])
```

Prueba un dropout desde 0.9 hasta 0.1. Explica los diferentes comportamientos observados y cómo impacta este valor

Respuesta:

Codificación:

Se ha realizado unas modificaciones al código para que podamos hacer un proceso de “hiperparametrización” del rate de la capa Dropout. Para ello hemos generado varias funciones:

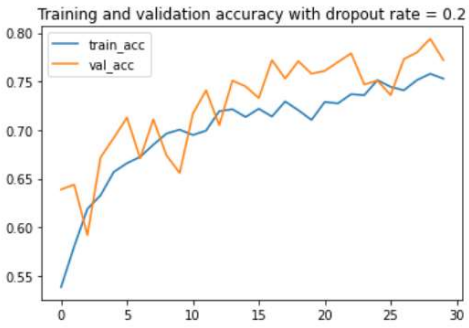
- **exec_CNN :** Función que encapsula la creación la compilación y el entrenamiento de la CNN. Como parámetro de entrada tiene el rate de la capa Dropout.
- **print_result:** Función que visualiza los resultados de cada entrenamiento con un rate dado. Se visualizan comparativamente loss y val_loss y accuracy y val_accuracy
- **Bucle del rate 0,1 al 0,9 saltando de 0,1 en 0,1.** Se realizan 9 llamadas a exec_CNN, para realizad entrenamientos un rate determinado y luego se visualizan resultados.

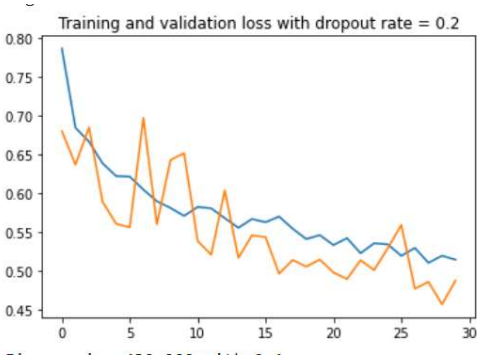
Conclusiones:

Con los resultados obtenidos se observa que para el rate=0,2 se obtienen los mejores resultados debido a:

- val_acc y tran_acc llegan en algunos momentos al 80%
- val_loss y train_loss van descendiendo progresivamente llegando a los menores valores registrados en torno al 45%

DropOut rate = 0.2 Scores

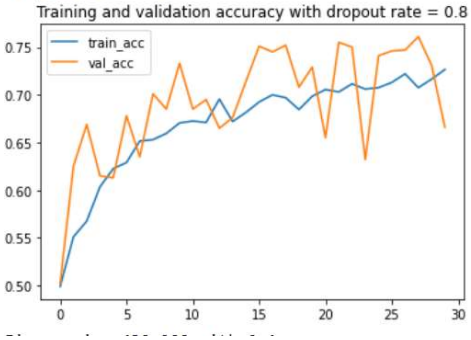


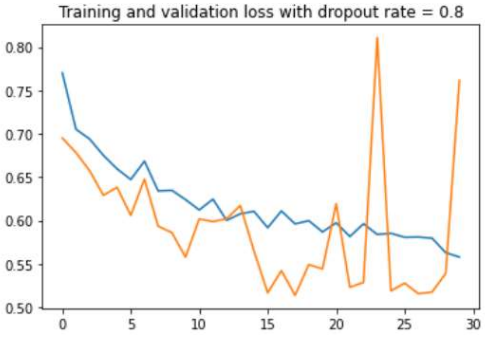


Los peores resultados se encuentran en torno al rate=0,9-0,8 donde:

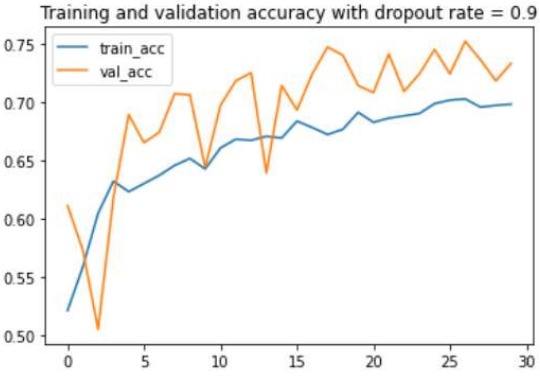
- val_acc y tran_acc no superan el 75%
- val_loss y train_loss llegan a picos de 0,9 en la función de perdida.

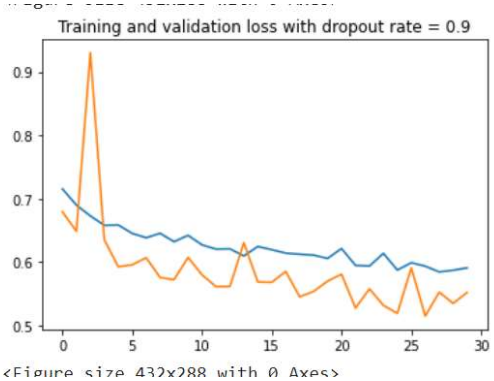
DropOut rate = 0.8 Scores





DropOut rate = 0.9 Scores





<Figure size 432x288 with 0 Axes>

3. SOBRE HOMBROS DE GIGANTES

3.1. [1 pts] TODO 1. Investiga acerca del Inception V3 y explica su topología y el concepto general. ¿Qué data base se ha utilizado para entrenarlo? ¿Cómo se entrenó?

Respuesta:

La red CNN Incetion_V3 es la tercera generación de las redes Inception. Basadas en el concepto de GoogleNet. Fue creada por Google y se presentó en el Imagine Recognition Challenge y explicada en el paper ***Rethinking the Inception Architecture for Computer Vision***. Las líneas principales de la arquitectura de Inception V3 es:

- Permitir una red más profunda de las que se venían teniendo, Inception-V3 tiene 47 capas.
- Evitar que el número de parámetros creciera y así hacerla más liviana en cuanto al procesamiento. Tiene menos de 25 millones de parámetros en contraposición de rivales como AlexNet que tiene 60 millones.

Esta CNN fue entrenada con la BBDD Imagenet (https://image-net.org/) que contiene 1,281,167 imágenes, clasificadas en más de 1000 clases. Alcanzó una exactitud aproximada del 78,1%

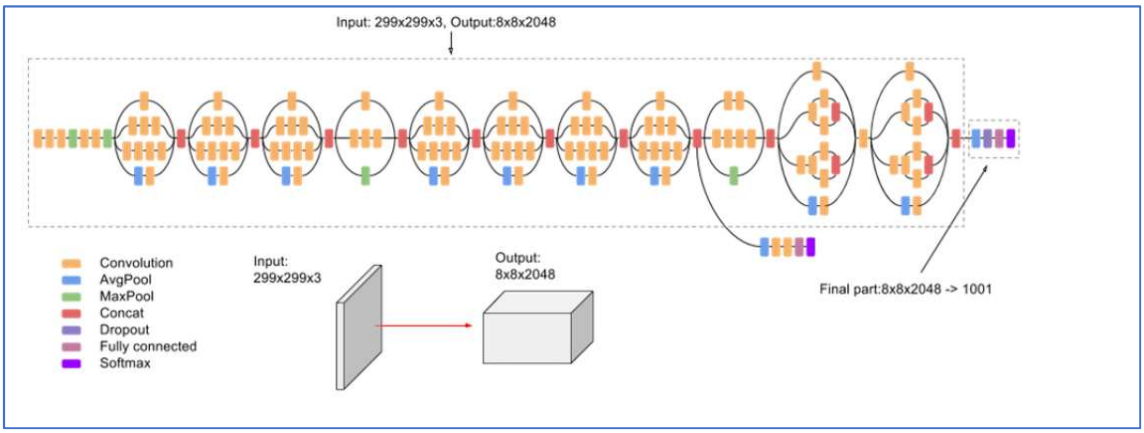


Figura 2: Arquitectura de la CNN Inception_V3 de Google.

Como en nuestro caso de la práctica se suele utilizar bastante en los casos de **Transfer Learning**, en el que utilizando su potencia y preentraniento, lo queremos aplicar en caso particulares, como el nuestro de detección de tipos de flores.
Para ello generaremos una red DNN o back forward para que sea la que clasifique nuestras clases, y la conectaremos a la última capa convulacional seguida de una flatten de Inception V3.

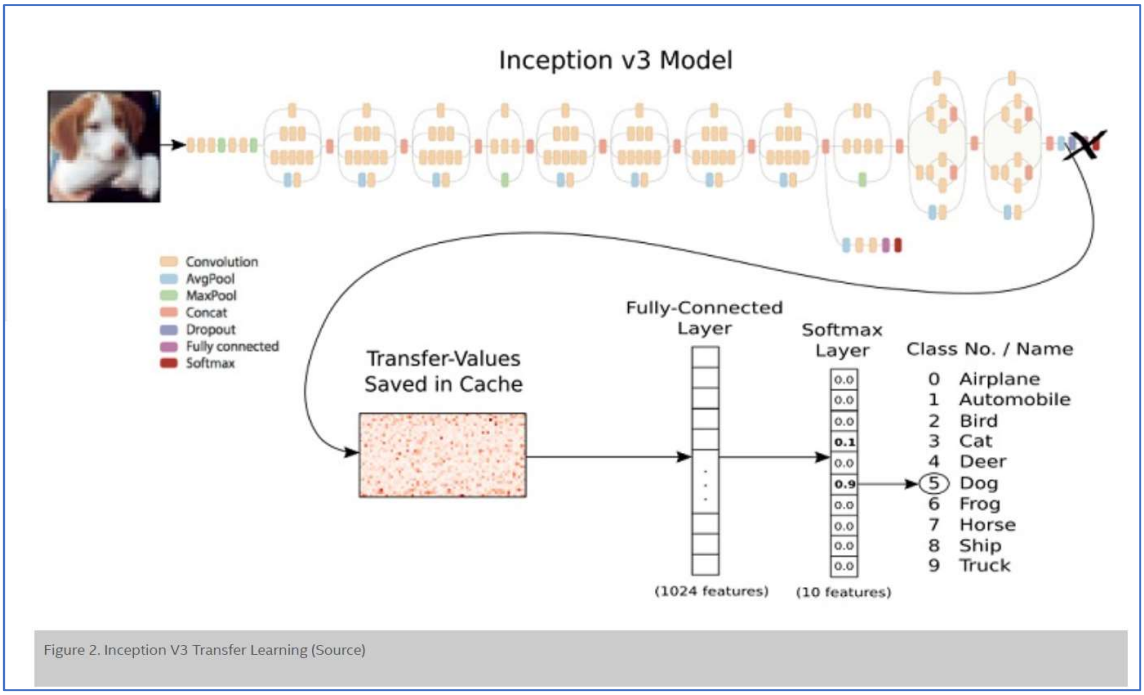


Figure 2. Inception V3 Transfer Learning (Source)

Un caso práctico de transfer learning de esta CNN ha sido el proyecto de detección de tipos de leucemia como el **Acute Myeloid & Lymphoblastic AI Research Project**

En cuanto a su arquitectura las principales ideas son:

- **Convoluciones factorizadas:** esto ayuda a reducir la eficiencia computacional ya que reduce el número de parámetros involucrados en una red. También controla la eficiencia de la red.
- **Convoluciones más pequeñas:** reemplazar las circunvoluciones más grandes con circunvoluciones más pequeñas definitivamente conduce a un entrenamiento más rápido. Digamos que un filtro de 5 × 5 tiene 25 parámetros; dos filtros de 3 × 3 que reemplazan una convolución de 5 × 5 tienen solo 18 (3 * 3 + 3 * 3) parámetros en su lugar.
- **Convoluciones asimétricas**: una convolución de 3 × 3 podría sustituirse por una convolución de 1 × 3 seguida de una convolución de 3 × 1. Si una convolución de 3 × 3 se reemplaza por una convolución de 2 × 2, el número de parámetros sería ligeramente mayor que la convolución asimétrica propuesta.
- **Clasificador auxiliar:** un clasificador auxiliar es una pequeña CNN insertada entre capas durante el entrenamiento, y la pérdida incurrida se suma a la pérdida de la red principal. En GoogLeNet se utilizaron clasificadores auxiliares para una red más profunda, mientras que en Inception v3 un clasificador auxiliar actúa como un regularizador.
- **Reducción del tamaño de la cuadrícula: la reducción del tamaño de la** cuadrícula generalmente se realiza mediante operaciones de agrupación. Sin embargo, para combatir los cuellos de botella del costo computacional, se propone una técnica más eficiente.

3.2. [1 pts] TODO 2. Inception V3 trabaja por defecto con imágenes de tamaño (299, 299, 3); sin embargo, nosotros queremos usar entradas de tamaño (150, 150, 3). Si la red ya está entrenada… ¿cómo es esto posible?

Respuesta:

Si es posible. Es posible debido a que si a la hora de crear el objeto de la Inception_V3, el parámetro **include_top lo igualamos a False** le permitimos trabajar con imágenes de menor tamaño de las que en un principio fue entrenada esta CNN. Eso si recomiendan que no sean menor de 75x75.

En ese enlace de **tensorflow** explican esta casuística.

Args	
include_top	Boolean, whether to include the fully-connected layer at the top, as the last layer of the network. Default to True.
weights	One of None (random initialization), imagenet (pre-training on ImageNet), or the path to the weights file to be loaded. Default to imagenet.
input_tensor	Optional Keras tensor (i.e. output of layers.Input()) to use as image input for the model. input_tensor is useful for sharing inputs between multiple different networks. Default to None.
input_shape	Optional shape tuple, only to be specified if include_top is False (otherwise the input shape has to be (299, 299, 3) (with channels_last data format) or (3, 299, 299) (with channels_first data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 75. E.g. (150, 150, 3) would be one valid value. input_shape will be ignored if the input_tensor is provided.

3.3. [2 pts] TODO 3. Vamos a cambiar la base de datos. ¿Trabajará bien Inception con otros objetos? Usa Inception V3 para mejorar los resultados de una pequeña CNN sobre una base de datos de flores. Completa el código en 04_cnn_template.ipynb para utilizar Inception V3 en esta base de datos de la misma forma que hicimos con cats & dogs

Respuesta:

Para poder trabajar con Inception V3 para el caso de las flores, se ha tenido que cambiar la configuración de la red, tanto de la parte de Input de la propia Inception V3, como de la DNN o back-forward que hemos tenido que construir como clasificador. Las modificaciones han sido:

Se ha tenido que adaptar el tamaño de la imagen a 100x100. Como ya vimos en otro apartado es posible trabar con tamaños más pequeños si al inicializar

Inception_V3 ponemos el parámetro **include_top=False** y en el parámetro **input_shape=(100,100,3)**

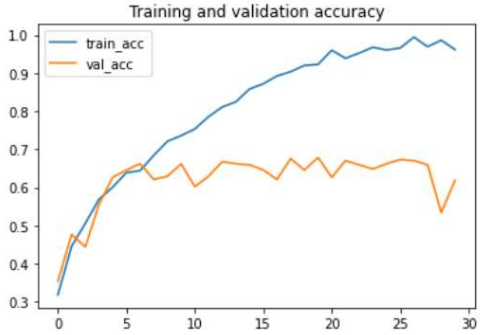
```
from tensorflow.keras.applications.inception_v3 import InceptionV3

local_weights_file = '/tmp/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'
pre_trained_model = InceptionV3(
    input_shape=(100, 100, 3), include_top=False, weights=None)
pre_trained_model.load_weights(local_weights_file)
```

También se ha tenido que cambiar la última capa Dense de la parte clasificadora. En este caso al ser multiclase y no binaria como en el caso de los cat&dogs. En este caso tenemos **5 clases,** por lo que la capa de salida la hemos configurado con una **softmax.**

En cuanto a los resultados en comparativa con la CNN que teníamos construida podemos comentar que se mejoran, notablemente. Tanto en el indicador accuracy tanto en test como en validation, así también en la función loss:

Los resultados de la pequeña CNN que hemos hecho son los siguientes:



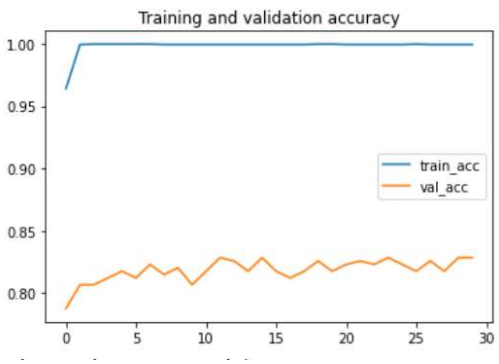
Los resultados de la Inception V3 sin en el fine tunning:

Se observa que al aplicar el modelo tiende al overfeating, ya que en los datos de entrenamiento llega a un accuracy de casi un 100% rápidamente y una función de

coste casi de 0. Mientras que validación no llega al 80% de accuracy y la función de pérdida da picos de 1.0

Los resultados de la Inception V3 con en el fine tunning:

Con el **fine tunning** el efecto overfeating es mucho más pronunciado que sin él, ya que en la 3 epoch en train ya llega al 100% y con un loss mínimo muy cercano a 0.



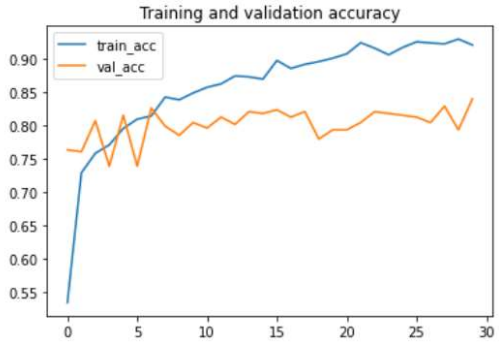
<Figure size 432x288 with 0 Axes>

He realizado una versión del **notebook_4_augmentation** en el que he metido unas layers de Keras de DataAugmentation. Siguiendo la estrategia de truncar **Inception_V3** en mixed7, pero haciéndolo de otra forma, los resultados parecen mucho mejores ya que debido a que no presenta tantos "síntomas" de tener overfeating. Por lo que parece que las capas metidas de **data augmentation** hacen su trabajo. Con **fine-tunning** mejoran los scores.

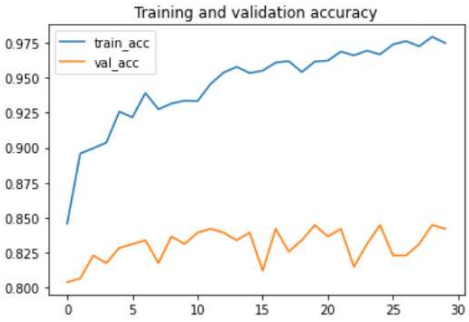
De esta forma se he montado la InceptionV3 truncada en mixed7

```
# Generamos un modelo IncepioV3 recortado a la capa mixed7.
# He cambiado un poco la extrategia ya que es modelo va a tener previamante unas capas de Data Augmentation
outputs = pre_trained_model.get_layer('mixed7').output
new_pretained_model=Model(pre_trained_model.input, outputs,name='InceptionV3_trunk_mixed7')
new_pretained_model.summary()
```

Las gráficas del modelo con **data augmentation** son las siguientes:



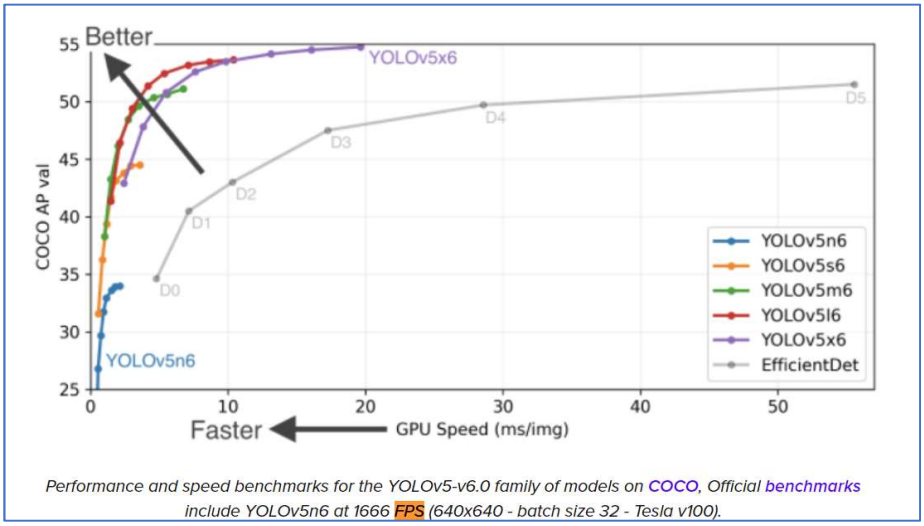
Las gráficas del modelo con **data augmentation y fine-tunning** son las siguientes:



4. Detección de Objetos. Redes YOLO

Son un nuevo tipo de redes CNN, utilizadas principalmente para la detección de objetos en tiempo real, por lo que son muy utilizadas en la detección de objetos en videos.

El nombre YOLO es un acrónimo de You Only Look Once y nos da la pista porque son redes con tan eficiente funcionamiento y es debido a que la imagen solo la visualizan una única vez. La reciente versión YOLO_V5 V6 es capaz de procesar hasta 1666 FPS (frame per second) y cuya versión nano puede correr en CPU o dispositivos móviles.



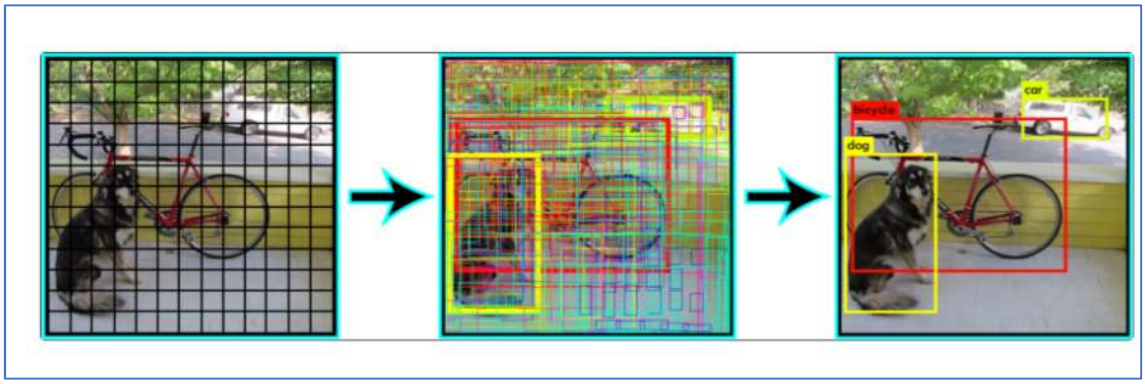
Performance and speed benchmarks for the YOLOv5-v6.0 family of models on COCO, Official benchmarks include YOLOv5n6 at 1666 FPS (640x640 - batch size 32 - Tesla v100).

YOLO ha tenido varias reléase o versiones:

- **Yolov1** (8 de junio de 2015): Solo miras una vez: Detección de objetos unificada en tiempo real
- **Yolov2** (25 de diciembre de 2016): YOLO9000: Mejor, más rápido, más fuerte
- **Yolov3** (8 de abril de 2018): YOLOv3: Una mejora incremental
- **Yolov4** (23 de abril de 2020): YOLOv4: Velocidad óptima y precisión de detección de objetos
- **Yolov5** (18 de mayo de 2020): repositorio de Github

A nivel general lo que hace una red YOLO en el procesado de las imágenes es dividirlas en cuadriculas de S x S y sobre estas cuadriculas busca “bounding boxes” con un nivel de certidumbre que en las primeras iteraciones será bajo. Posteriormente elimina las que están por debajo de un límite. A las cajas restantes se les aplica un paso llamado

“non-max supression” que sirve para eliminar objetos que fueron detectados por duplicado y así dejar el más exacto como ocurre en la imagen final de la derecha.



Hablando ya de la última versión de YOLO_V5 liberada en junio de 2021 por Ultralitics, cuyo CEO es Glenn Jocher.

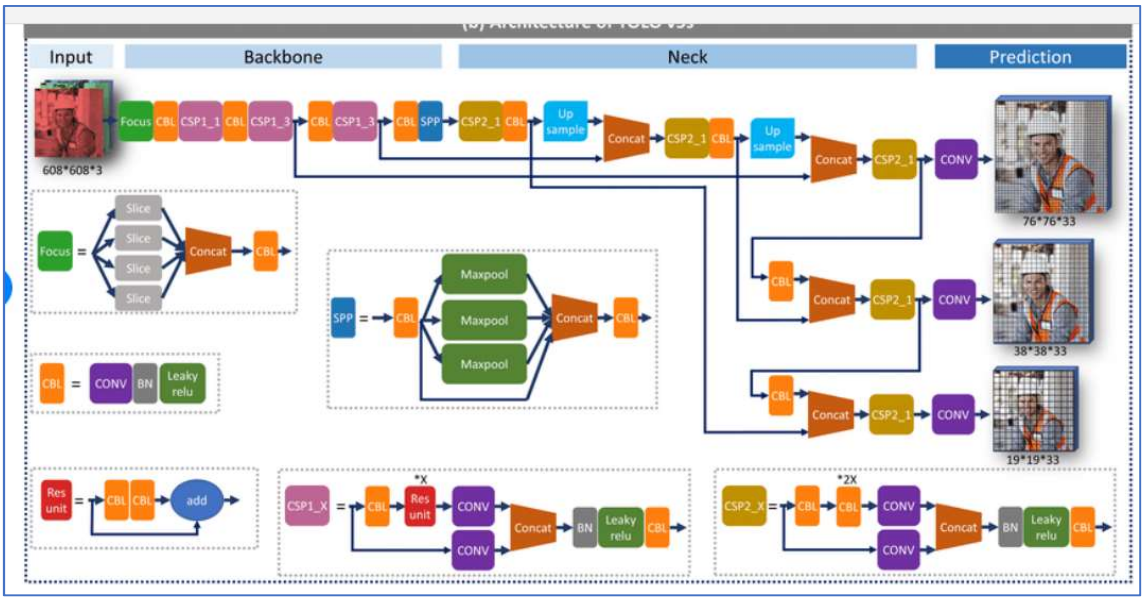
A nivel de arquitectura y hablando de la ultima versión la YOLO_V5, esta tiene 26 capas, 24 capas propiamente de convolución y 2 capas full conected.

Todas estas capas se pueden agrupar en 3 bloques:

- **Backbone** : una red neuronal convolucional que agrega y formas características de imagen en diferentes granularidades.

- **Neck :** una serie de capas para mezclar y combinar características de la imagen para pasarlas a la predicción.

- **Head :** consume características del cuello y realiza pasos de predicción de clase y caja.



Como características técnicas principales de la nueva versión de YOLO_V5 con respecto a la anterior YOLO_V4:

- La V5 está basada en Pytorch, por el contrario de la V4 esta basada en DrakNet principalmente escrita en C.

- Los ficheros de configuración del modelo en YOLO_V5 están escritos en YAML y podríamos configurar a nuestro interés modificando los valores del fichero. En V4 son cfg. El yaml, Tienen este aspecto:

```yaml
# parameters
nc: 80  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple

# anchors
anchors:
  - [116,90, 156,198, 373,326]  # P5/32
  - [30,61, 62,45, 59,119]  # P4/16
  - [10,13, 16,30, 33,23]  # P3/8

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, BottleneckCSP, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
  ]

# YOLOv5 head
head:
  [[-1, 3, BottleneckCSP, [1024, False]],  # 9

```

Por último, para el ejemplo que vamos a probar, he utilizado la plataforma Robloflow, para lo cual me he generado un usuario, y creado dos pequeños dataset con las piezas de ajedrez y otro con los cascos de obra que llevan unas personas. El notebook está basado en ejemplos que me he encontrado en Kaggle. El repositorio y los fuentes son bajados de la distribución de YOLO_V5 de Ultralytics https://github.com/ultralytics/yolov5.

El **notebook Polvorinos_Barrio_JuanPedro_05_cnn_tamplate_YOLO.ypnb** es el que tiene el dataset de ajedrez y los resultados han sido un poco pobres, ya que todo lo identifica como el peón negro. El dataset es solo de 28 imágenes,

El **notebook Polvorinos_Barrio_JuanPedro_05_cnn_tamplate_YOLO_v2.ypnb** es el que tiene el dataset de los cascos de obra y personas y los resultados han sido mejores. También es cierto que el dataset es mayor y las clases o label son menos.

El indicador que nos va a decir lo bien o mal que identifica el modelo es el mAP (mean average precisión)