Universidad Francisco de Vitoria
UFV Madrid

Asignatura:

Inteligencia Artificial II

Título del documento:

Laboratorio 03

Preparado por:

Nombre	Gonzalo Monedero Nicolás Salgado Iñigo Vázquez Marcos Molina	04/04/2022

1 Índice

1 Índice 2
2 Introducción 4
3 Práctica 1 6
3.1 Introducción 6
3.2 Estudio Redes 7
3.2.1 Red 1 8
3.2.1.1 Descripción 8
3.2.2 Red 2 9
3.2.2.1 Descripción 9
3.2.3 Red 3 10
3.2.3.1 Descripción 10
3.2.4 Red 4 11
3.2.4.1 Descripción 11
3.2.5 Red 5 12
3.2.5.1 Descripción 12
3.2.6 Red 6 13
3.2.6.1 Descripción 13
3.3 Comparación de las redes 14
3.3.1 Redes sin dropout 14
3.3.2 Redes con dropout 15
3.3.2.1 Arquitectura red 3 15
3.3.2.2 Arquitectura red 4 16
3.3.3 Redes con cambio MLP 17
3.4 Conclusión de la mejor red 18
3.5 Estudio del learning rate 19
3.6 Pruebas de red 20
3.7 Augmentation 22
4 Práctica 2 24
4.1 Introducción 24
4.2 Estudio Redes 24
4.2.1 Red 1 25
4.2.1.1 Descripción 25
4.2.2 Red 2 26
4.2.2.1 Descripción 26
4.2.3 Red 3 27
4.2.3.1 Descripción 27
4.2.4 Red 4 28
4.2.4.1 Descripción 28
4.2.5 Red 5 29
4.2.5.1 Descripción 29
4.2.6 Red 6 30

4.2.6.1 Descripción _____ 30
4.3 Conclusión de la mejor red _____ 31
4.4 Estudio del learning rate _____ 32
4.5 Pruebas de red _____ 32
Conclusión _____ 34

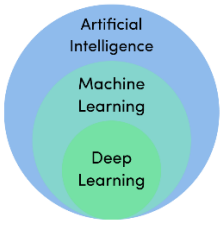
2 Introducción

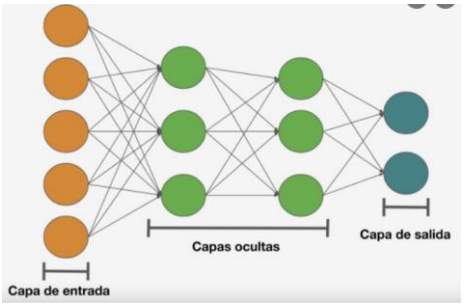
En esta práctica trataremos de desarrollar de forma autónoma distintas implementaciones de redes neuronales profundas de aprendizaje supervisado.

Pero antes que nada debemos aclarar qué es una red neuronal profunda lo que comúnmente se conoce como *Deep Learning.*

¿Qué es Deep Learning?

Una de las mejores definiciones de Deep Learning es: Área del Machine Learning que utiliza diferentes algoritmos de aprendizaje automático para modelar abstracciones de datos de alto nivel usando arquitecturas jerárquicas llamadas Redes Neuronales Profundas (DNN).





Este a su vez se basa en modelos computacionales que exhiben características similares a las del neocórtex, ya que podemos afirmar que el cerebro humano es una arquitectura neuronal profunda. Cuando hablamos de profundidad estamos hablando del camino más largo desde un nodo de entrada hasta el nodo de salida, lo que también es el nº de capas ocultas más la capa de salida ya que la de entrada no hace realmente nada más que enviara a la primera capa oculta los datos de entrada.

Uno de los principales problemas de este, es la profundidad y el problema de la difusión de gradientes, en el cual las ultimas capas desde la salida no cuentan con casi error mientras que las primeras se ajustan a la perfección, queriendo decir esto que las neuronas más profundas no aprenden casi por cada interacción.

Otros de los problemas asociados al DL supervisado que es el que vamos a realizar en esta práctica es que en muchas ocasiones los datos necesarios son difíciles de obtener y debido a esto, entrenar con datos insuficientes nos lleva al overfiting. No solo encontramos este sino también el problema de los óptimos locales.

Para estos problemas surgen diversas soluciones entre las cuales destacamos la de Hinton en dos pasos, donde en el 1er paso se pre-entrenan todas las capas de forma no supervisada y después se entrena de forma supervisada.

Una vez introducido esto, debemos aclarar que es el modelo DNN y de donde proviene ya que es el modelo para seguir en la práctica.
Este modelo proviene de un tipo de red llamada red convolucional la cual como su nombre indica, se basa en la convolución.

¿Qué es la convolución?

Operador matemático que transforma dos funciones f y g de una sola variable en una nueva función $f*g$ que representa la magnitud en la que se superponen f y una versión trasladada e invertida de g.

En nuestro caso trataremos con imágenes, lo que significa que cada píxel es un array con RGB, de esta forma a estas imágenes les aplicaremos filtros y transformaciones que convertirán nuestra imagen en otra que nos dará información de interés para el aprendizaje como puede ser detectar bordes, figuras...

De este modo llegamos a la finalidad de esta práctica, en la cual aprenderemos el funcionamiento completo de una red convolucional, que es el Deep Learning, sus principales problemas y las soluciones a estos y aprender a diseñar un modelo CNN el cual no sufra de overfiting.

3 Práctica 1

3.1 Introducción

En estar 1ª Practica debemos crear una red convolucional (CNN) la cual clasifique las distintas imágenes de forma supervisada.

Nos presentan un dataset de 4Gb de imágenes de 640x480 con sus etiquetas correspondientes:

- c0: safe driving
- c1: texting - right
- c2: talking on the phone - right
- c3: texting - left
- c4: talking on the phone - left
- c5: operating the radio
- c6: drinking
- c7: reaching behind
- c8: hair and makeup
- c9: talking to Passenger

El objetivo de este apartado es identificar la acción de un conductor mediante una red CNN entrenada que nos muestre las probabilidades de lo que está realizando el conductor.

Los pasos que deberemos seguir para crear esta red son:

1. Leer de forma correcta las imágenes.
2. Separar cada imagen en pixeles
3. Separar el conjunto de datos de entrenamiento y prueba
4. Creamos más datos para el dataset con augmentación
5. Normalizamos los datos
6. Creamos la arquitectura (Entrada Capa de conclusión 1,2,3+Salida)
7. Crear filtros con kernel(pesos) aleatorios.
8. Creamos el entrenamiento
9. Creamos la función que nos indique si la solución ha sido la óptima, en caso contrario, modificamos los kernel.

3.2 Estudio Redes

En el siguiente apartado realizaremos un estudio de redes neuronales en base a valores como el número de capas de convolución, el tamaño del MLP, el dropout… El objetivo principal de este estudio es saber como afecta cada variación a las redes, de tal manera que tras el estudio podamos afirmar cual es la mejor red, la cual usaremos para posteriormente la prueba de imágenes. Con esta red final dibujaremos su arquitectura de red.

Es por ello por lo que a continuación veremos primero una descripción de como es la arquitectura de cada red y cuales son sus gráficas de loss y accuracy tanto para el conjunto de entrenamiento como para el conjunto de validación.

Hay que matizar que todas las redes se entrenan con un criterio de parada en base al loss. Cuando el loss es menor a 0.2, la red deja de entrenar.

Es por ello por lo que primero veremos 4 redes con diferente arquitectura, que se estudian con y sin dropout para ver su influencia. Por último, estudiaremos dos redes adicionales con misma arquitectura, pero con diferente tamaño de MLP para ver también como influye esto en la calidad de la red.

Para finalizar un esquema de que veremos ver:

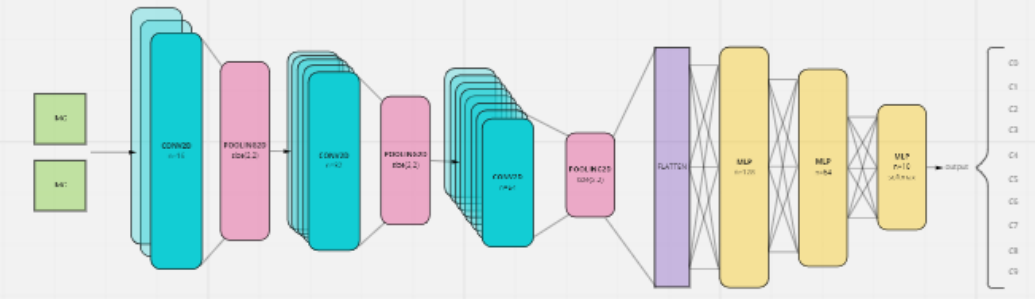
- En las cuatro primeras redes la influencia del número de capas de convolución y como afecta el dropout
- En las dos últimas redes la influencia del tamaño del MLP

3.2.1 Red 1

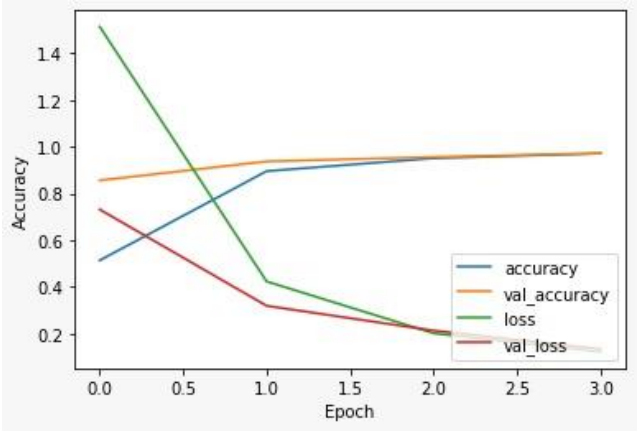
3.2.1.1 Descripción

En la primera red podemos ver la siguiente estructura. Tres capas de convolución con su respectivo pooling cada una. Por último, un MPL de 128 y 64 neuronas respectivamente.

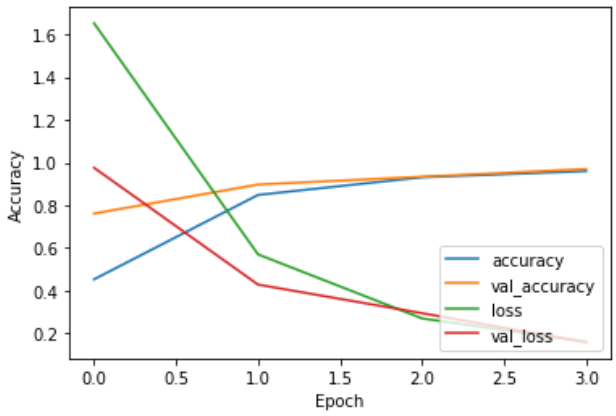
Cuando añadimos el dropout, se añade en cada capa de convolución.



```
model.add(Conv2D(filters=n_conv1, kernel_size=(
    3, 3), padding='same', activation='relu'))
# Capas convolucionales
model.add(MaxPooling2D())
#   >este bloque se puede seguir añadiendo, quiza con menos neuronas,
# (3,3) es mucho se recomienda unsar 1x1 cuando las img no son mayores
model.add(Conv2D(n_conv2, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2)) # dropout

model.add(MaxPooling2D())
# model.add(Conv2D(n_conv3,(3,3),activation='relu',padding='same'))
model.add(Conv2D(n_conv3, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
model.add(Dropout(0.2))

# Capa fully-connected - MLP
model.add(Flatten())
model.add(Dense(128, activation='relu'))  # red fully-connected
model.add(Dense(64, activation='relu'))  # red fully-connected
```



Sin dropout



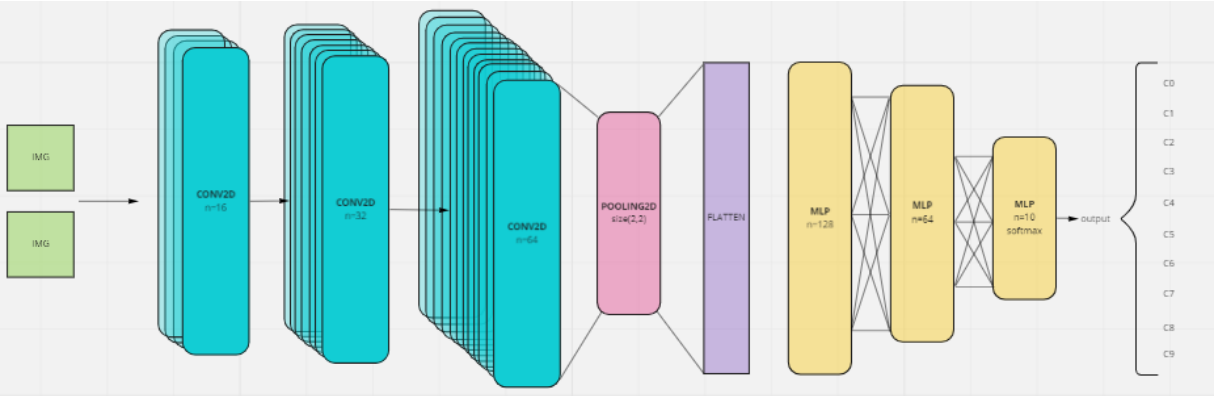
Con dropout

3.2.2 Red 2

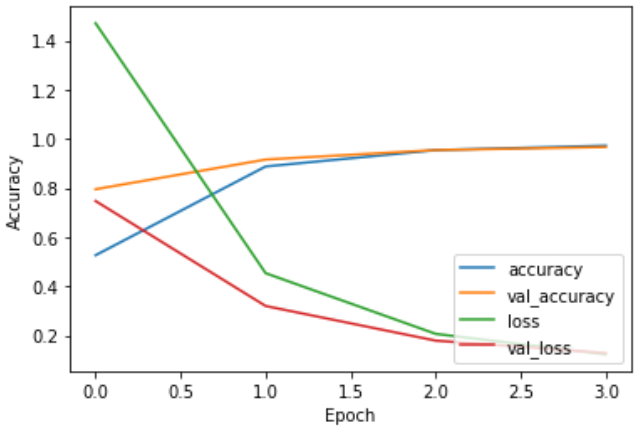
3.2.2.1 Descripción

En la segunda red se dan 3 capas de convolución seguidas con un único pooling al final. Por último, un MPL de 128 y 64 neuronas respectivamente.

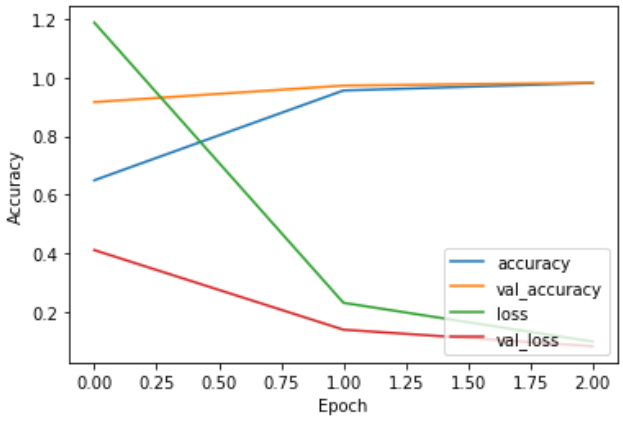
Cuando añadimos el dropout, se añade al final del pooling.




```
model.add(Conv2D(filters=n_conv1, kernel_size=(3, 3), padding='same', activation='relu'))
model.add(Conv2D(n_conv2, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(n_conv3, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D()) #model.add(MaxPooling2D(pool_size=(2,2)))
# model.add(Dropout(0.2))

# Capa fully-connected - MLP
model.add(Flatten())
model.add(Dense(128, activation='relu'))  # red fully-connected
model.add(Dense(64, activation='relu'))  # red fully-connected
```



Sin dropout



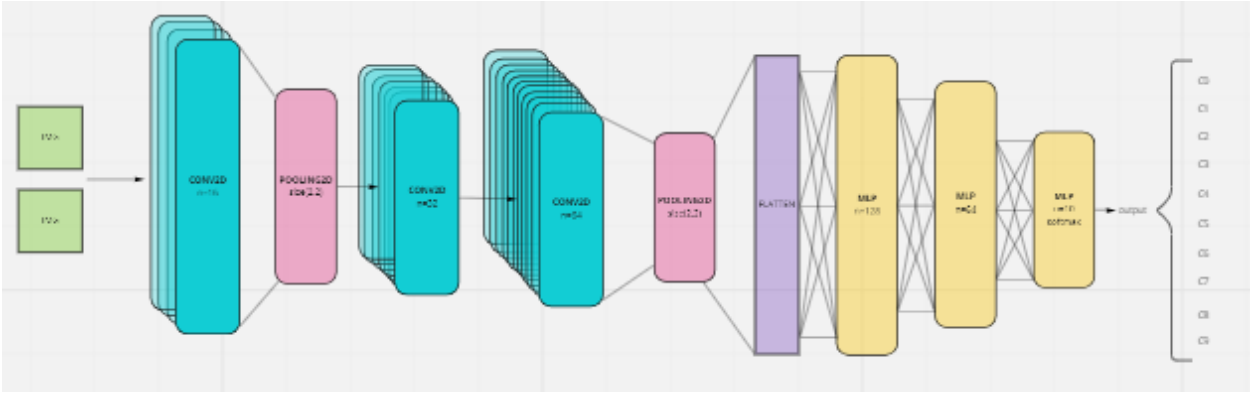
Con dropout

3.2.3 Red 3

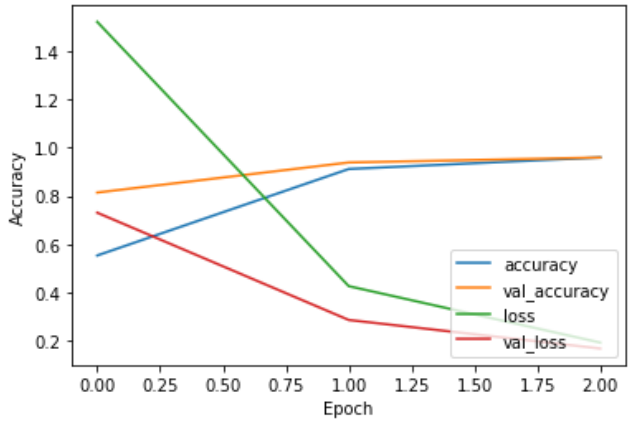
3.2.3.1 Descripción

En la tercera red existen tres capas de convolución, con un pooling en la 1º y 3º capa de convolución. Por último, un MPL de 128 y 64 neuronas respectivamente.

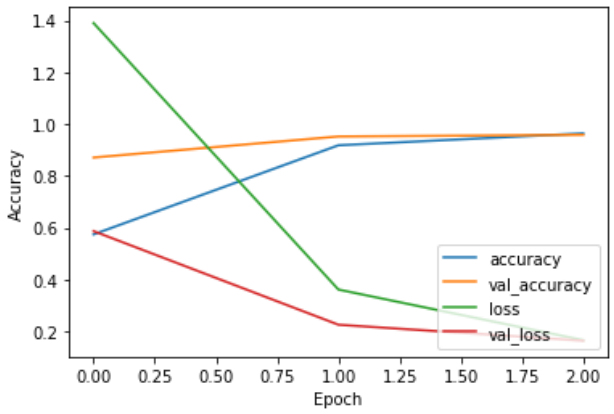
Cuando añadimos el dropout, se añade al final del pooling.



```
model.add(Conv2D(filters=n_conv1, kernel_size=(3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D())
# capas convolucionales concatenadas
model.add(Conv2D(n_conv2, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(n_conv3, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D()) #model.add(MaxPooling2D(pool_size=(2,2)))
# model.add(Dropout(0.2))

# Capa fully-connected - MLP
model.add(Flatten())
model.add(Dense(128, activation='relu'))  # red fully-connected
model.add(Dense(64, activation='relu'))  # red fully-connected
```



Sin dropout



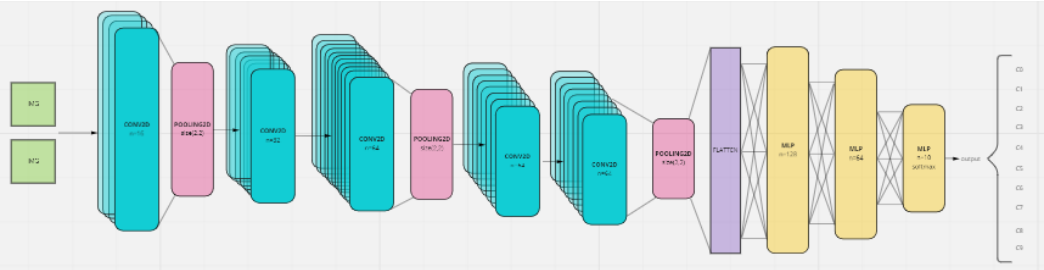
Con dropout

3.2.4 Red 4

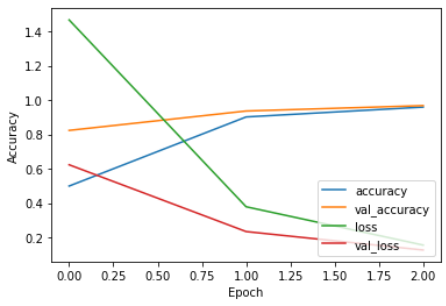
3.2.4.1 Descripción

En la cuarta red existen cinco capas de convolución, con un pooling en la 1º, 3º y 5º capa de convolución. Por último, un MPL de 128 y 64 neuronas respectivamente.

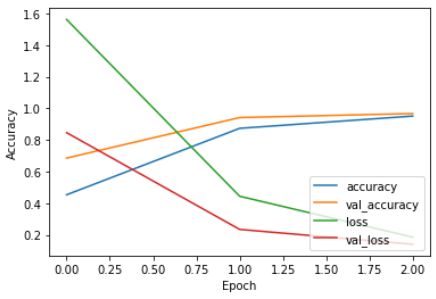
Cuando añadimos el dropout, se añade al final del pooling.



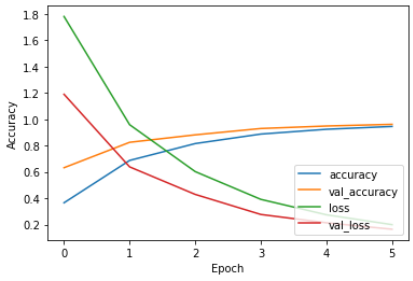
```
model.add(Conv2D(filters=n_conv1, kernel_size=(3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D())
model.add(Conv2D(n_conv2, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(n_conv3, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D()) #model.add(MaxPooling2D(pool_size=(2,2)))
# model.add(Dropout(0.2))
model.add(Conv2D(n_conv3*2, (3, 3), activation='relu', padding='same'))    # lo multiplico
model.add(Conv2D(n_conv3*4, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
# Capa fully-connected - MLP
model.add(Flatten())
model.add(Dense(128, activation='relu'))  # red fully-connected
model.add(Dense(64, activation='relu'))  # red fully-connected
```



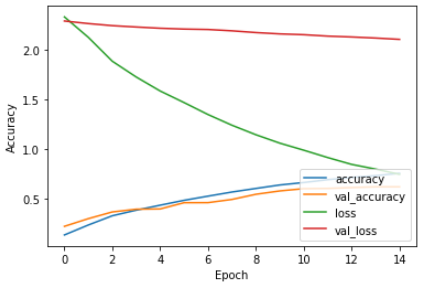
Sin dropout



Dropout 0.2



Dropout 0.5



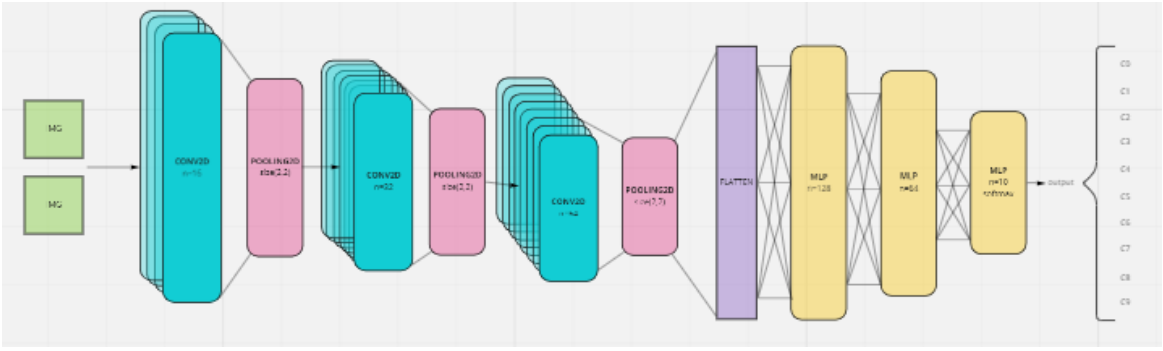
Dropout 0.9

3.2.5 Red 5

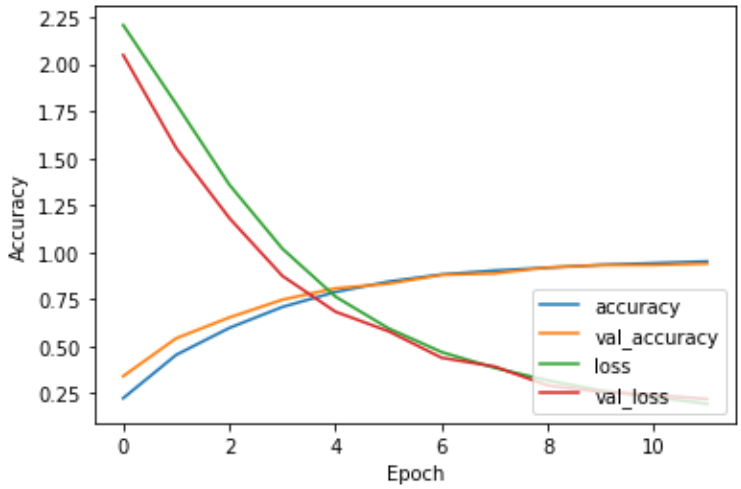
3.2.5.1 Descripción

En la primera red podemos ver la siguiente estructura. Tres capas de convolución con su respectivo pooling cada una. Por último, un MPL de 32 y 16 neuronas respectivamente.

Cuando añadimos el dropout, se añade en cada capa de convolución.



```
model.add(Conv2D(filters=n_conv1, kernel_size=(3, 3), padding='same', activation='relu'))
# Capas convolucionales
model.add(MaxPooling2D())
#   >este bloque se puede seguir añadiendo, quiza con menos neuronas, o menos capas convolu
# (3,3) es mucho se recomienda unsar 1x1 cuando las img no son mayores de 128x128 la nuestr
model.add(Conv2D(n_conv2, (3, 3), activation='relu', padding='same'))
# model.add(Dropout(0.2)) # dropout

model.add(MaxPooling2D())
# model.add(Conv2D(n_conv3,(3,3),activation='relu',padding='same'))
model.add(Conv2D(n_conv3, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
# model.add(Dropout(0.2))

# Capa fully-connected - MLP
model.add(Flatten())
model.add(Dense(32, activation='relu'))  # red fully-connected
model.add(Dense(32/2, activation='relu'))  # red fully-connected
```



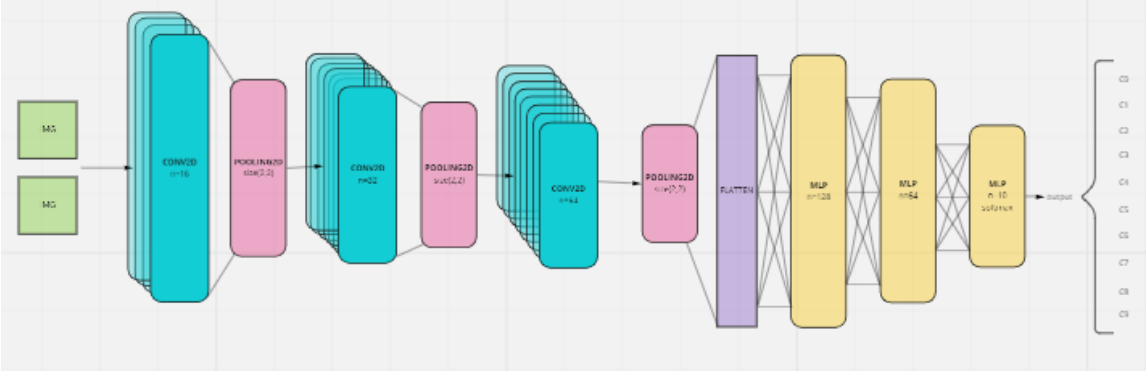
32 y 16 neuronas MLP

3.2.6 Red 6

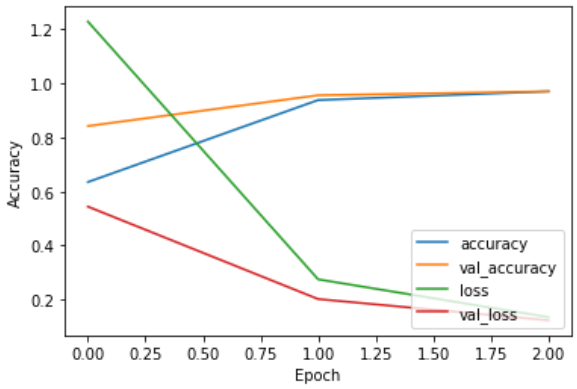
3.2.6.1 Descripción

En la primera red podemos ver la siguiente estructura. Tres capas de convolución con su respectivo pooling cada una. Por último, un MPL de 1024 y 512 neuronas respectivamente.

Cuando añadimos el dropout, se añade en cada capa de convolución.



```
model.add(Conv2D(filters=n_conv1, kernel_size=(3, 3), padding='same', activation='relu'))
# Capas convolucionales
model.add(MaxPooling2D())
#   >este bloque se puede seguir añadiendo, quiza con menos neuronas, o menos capas convol
# (3,3) es mucho se recomienda unsar 1x1 cuando las img no son mayores de 128x128 la nuest
model.add(Conv2D(n_conv2, (3, 3), activation='relu', padding='same'))
# model.add(Dropout(0.2)) # dropout

model.add(MaxPooling2D())
# model.add(Conv2D(n_conv3,(3,3),activation='relu',padding='same'))
model.add(Conv2D(n_conv3, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
# model.add(Dropout(0.2))

# Capa fully-connected - MLP
model.add(Flatten())
model.add(Dense(1024, activation='relu'))  # red fully-connected
model.add(Dense(1024/2, activation='relu'))  # red fully-connected
```

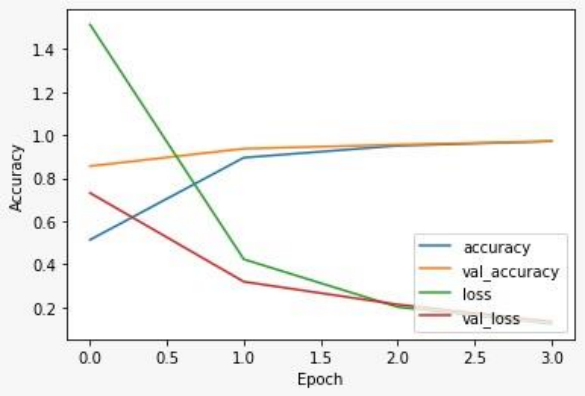


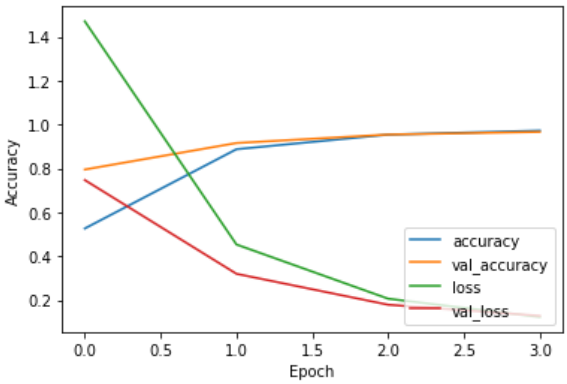
1024 y 512 neuronas MLP

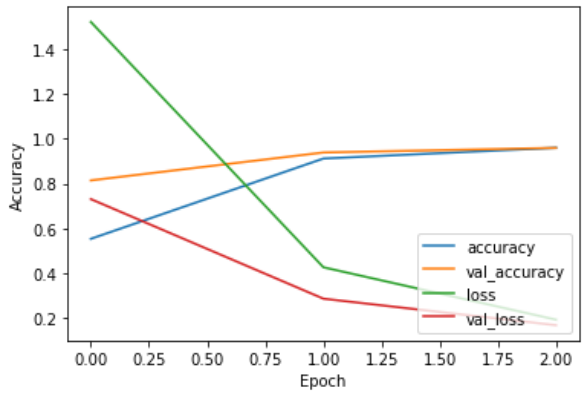
3.3 Comparación de las redes

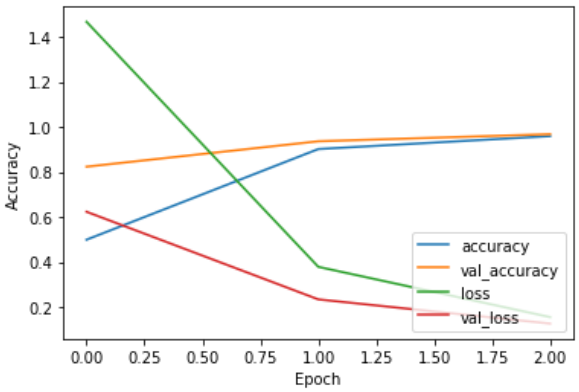
3.3.1 Redes sin dropout

En la siguiente tabla podemos ver las cuatro redes, viendo su arquitectura y sus gráficas.







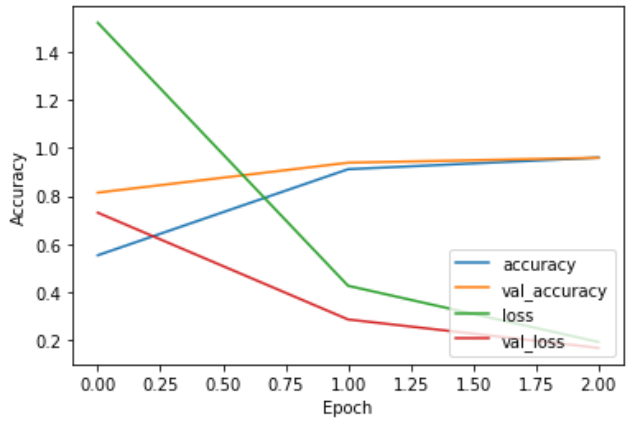


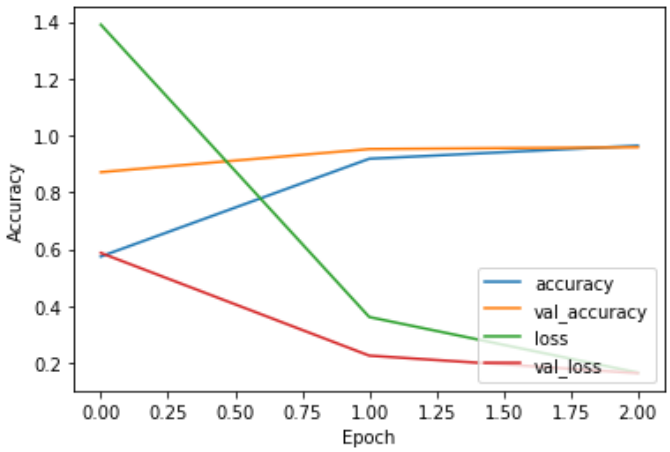
Podemos ver en las dos últimas gráficas como son capaces de maximizar el accuracy en un epoch menos. Estas dos arquitecturas comparten que tienen más pooling que las demás. La arquitectura tres tiene **3 capas de convolución y 2 pooling,** por otro lado, la cuatro, tiene **5 capas de convolución y 3 pooling.** Esto nos lleva a la conclusión de que la importancia es los pooling.

3.3.2 Redes con dropout

A continuación, compararemos las mejores gráficas (la gráfica de las arquitecturas 3 y 4) con y sin dropout, para ver cuán importante es el dropout para un correcto funcionamiento.

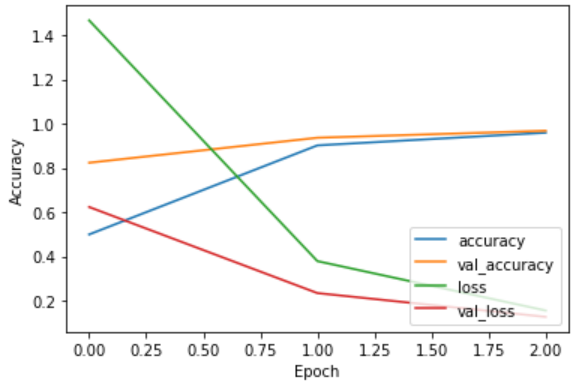
3.3.2.1 Arquitectura red 3

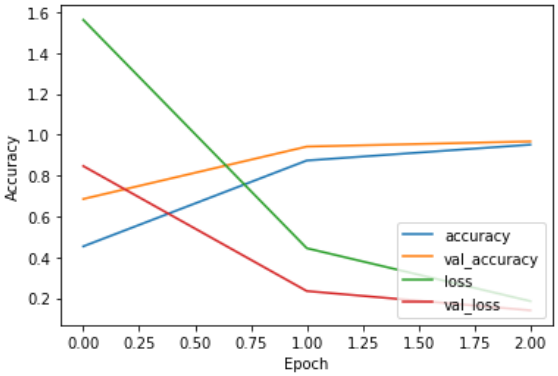


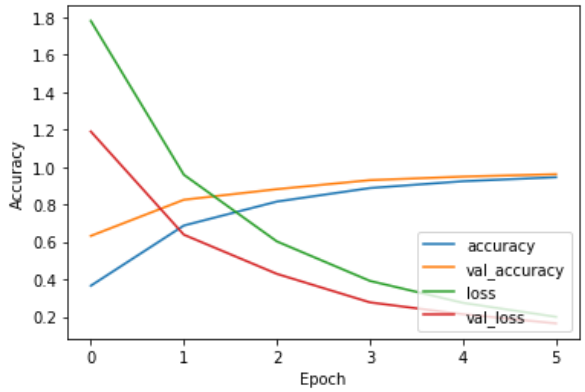


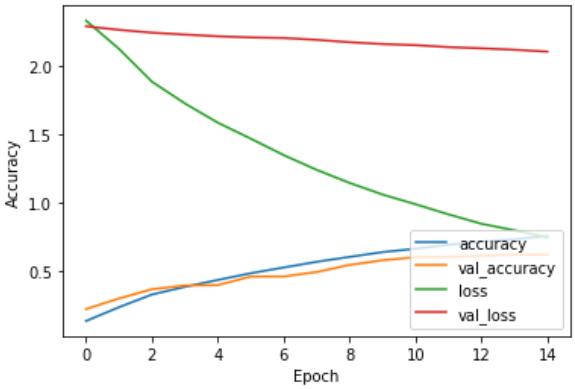
Podemos ver como con el dropout se produce un ligero cambio o apenas nulo. Las gráficas no nos muestran entonces que el dropout signifique un cambio sustancial en los valores de entrenamiento y validación, pero aun así lo seguiremos añadiendo. El porque de esto es que el dropout nos permite usar un $x\%$ menos de las neuronas de nuestra red, permitiendo así que nuestra red no memorice. De esta manera podremos entrenar la red con una mayor calidad pese a que las gráficas no lo muestren visiblemente.

3.3.2.2 Arquitectura red 4



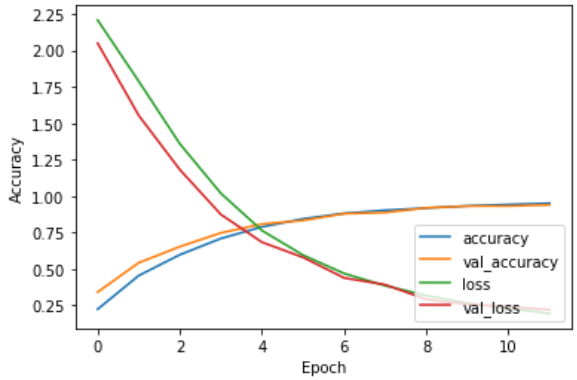


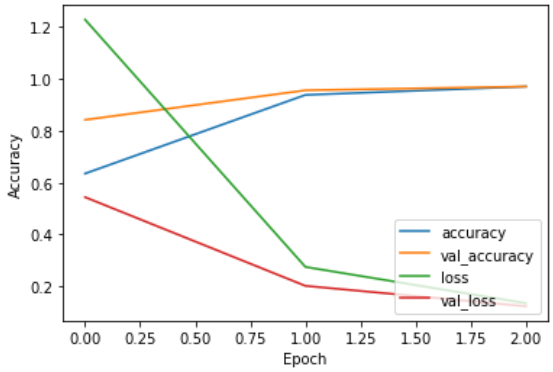




Podemos ver como cuanto más subimos el dropout a la red, tarda mucho más en entrenarse y tiene un entrenamiento de peor calidad, viendo como los valores de loss son más altos y de accuracy más bajos. De esta manera entendemos que el dropout es importante, pero bajo unos valores razonables.

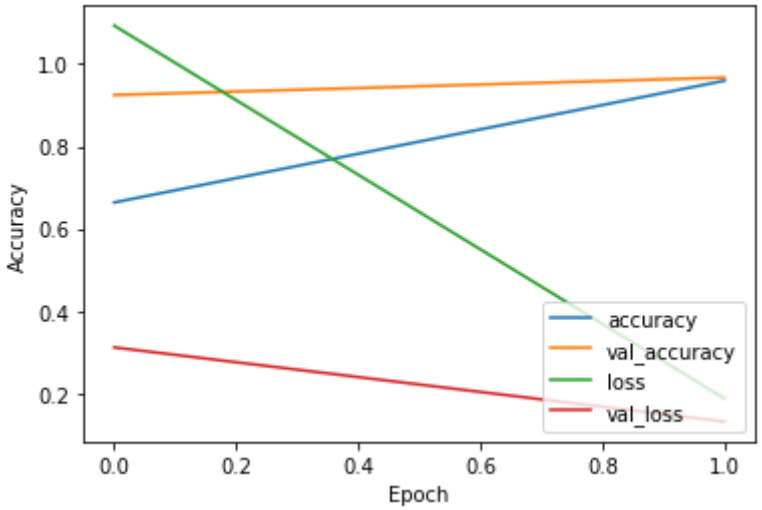
3.3.3 Redes con cambio MLP





En este caso vemos como el tamaño del MLP influye considerablemente en la calidad de la red, viendo como a mayor MLP, mejor predicción realiza nuestra red. Es por ello por lo que siempre y cuando nuestra capacidad computacional nos lo permita, a mayor MLP mayor calidad.

Hay que matizar que hemos querido probar con MLPs mayores, como de 2^{12} neuronas, pero nuestros ordenadores no nos han permitido entrenarlos, puesto que la memoria se veía sobrecargada. A continuación, se adjunta una foto de una red entrenada con 2^{12} que conseguimos entrenar una vez, viendo como realmente no tiene un cambio tan significativo como para que nos merezca la pena sacrificar nuestra capacidad de computación por un ligero cambio.

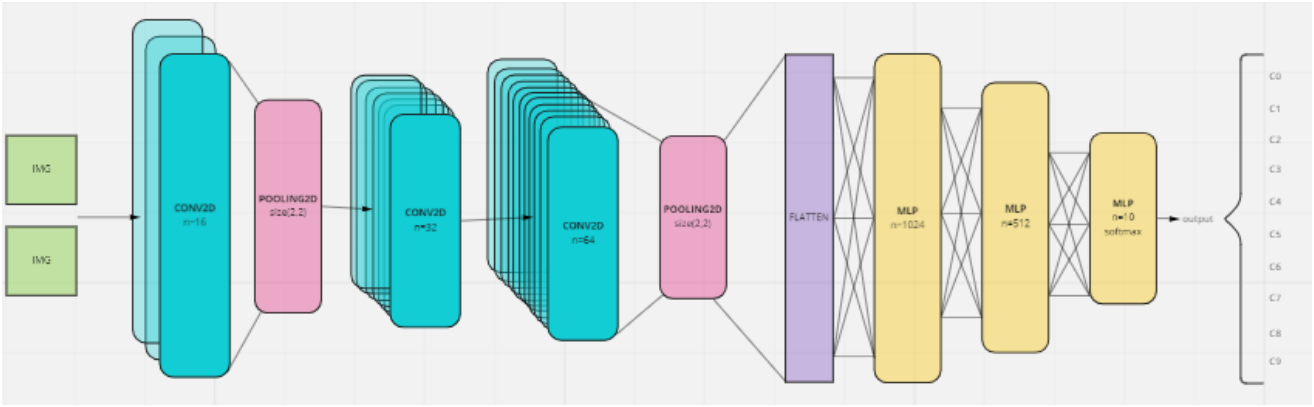


3.4 Conclusión de la mejor red

En los anteriores apartados hemos llegado a tres conclusiones:

- El pooling es muy necesario
- El dropout es importante que esté, pero bajo sin excederse
- A mayor MLP dentro de unos valores razonables, mejor funcionará nuestra red.

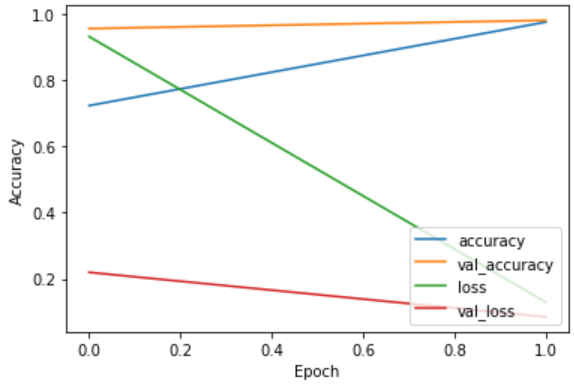
Por tanto, sabiendo las tres conclusiones, podemos llegar a la solución de nuestra red idónea para esta práctica. **Una red con tres capas de convolución, con un pooling en la 1º y 3º capa de convolución y dropout de 0.2. Por último, un MPL de 1024 y 512 neuronas respectivamente.**



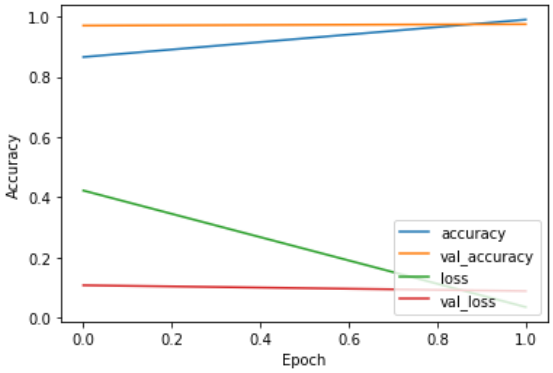
3.5 Estudio del learning rate

Una vez tenemos la arquitectura de la red estudiada, el último valor que varía nuestra calidad de red es el learning rate. Es por ello por lo que a continuación procedemos a realizar un estudio exhaustivo de cuál es el learning rate más adecuado para nuestra red.

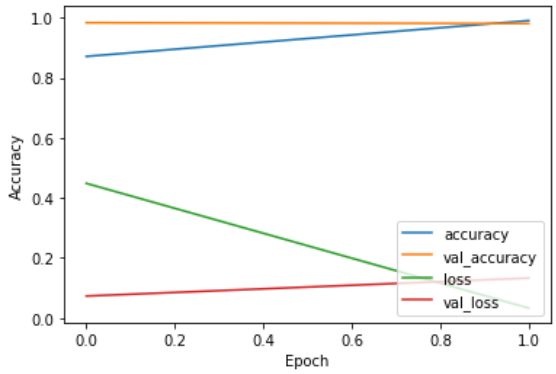
Es por ello por lo que en las siguientes cuatro gráficas probaremos los learning rates respectivos: 0.00001,0.0001,0.001,0.01 y veremos cual es el más adecuado.



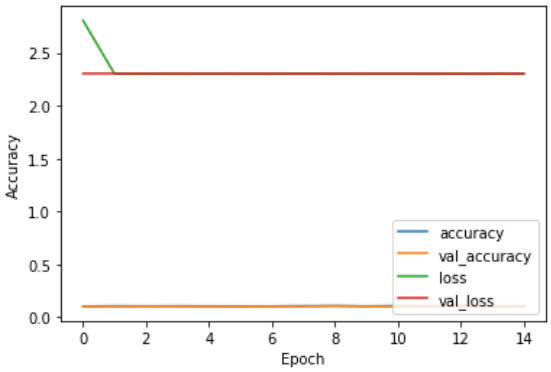
Learning rate 0.00001



Learning rate 0.0001



Learning rate 0.001



Learning rate de 0.01

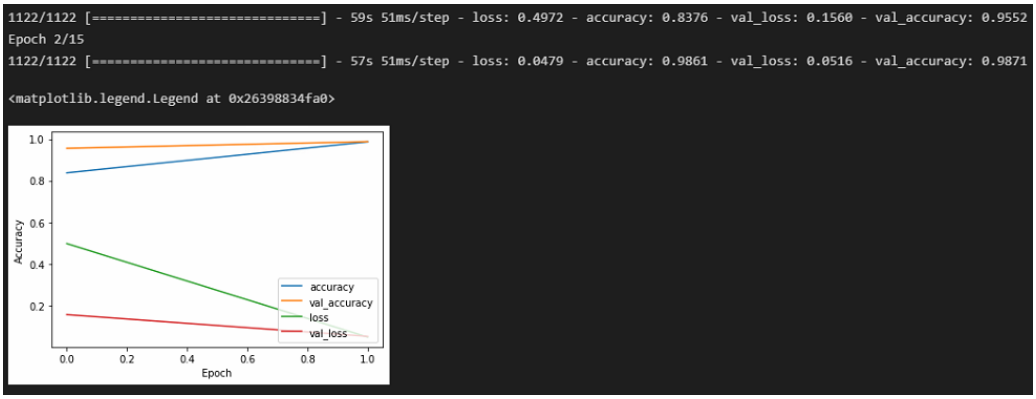
Podemos ver entonces como el loss se reduce significativamente y el accuracy se maximiza con los learning rate de 0.0001 *y* 0.001, mientras que el de 0.00001 es estándar y el de 0.01 da valores deplorables. Viendo más afondo los valores de resultado vemos como el learning rate de 0.0001 es ligeramente mejor, y por tanto usaremos ese.

3.6 Pruebas de red

Finalmente, ya sabemos cuál es la arquitectura idónea de nuestra red, una red con tres capas de convolución, con un pooling en la 1º y 3º capa de convolución y dropout de 0.2. Por último, un MPL de 1024 y 512 neuronas respectivamente junto a su learning rate de 0.0001.

Sabiendo esto, queda la parte final, la de clasificar el conjunto de imágenes de prueba, recogiendo en la memoria la salida obtenida, además de un CSV para todas las imágenes de prueba.

Es por ello por lo que primero entrenamos nuestra red, que como resultado nos da los siguientes valores de accuracy en dos epoch:



Además, también realizamos la evaluación de nuestra red con los datos de entrenamiento y de validación, sacándonos el siguiente resultado:

```
train_loss, train_acc = Evaluar(model,train)
val_loss, val_acc = Evaluar(model,validation)
✓ 19.3s

1122/1122 [==========] - 15s 14ms/step - loss: 0.0224 - accuracy: 0.9928
281/281 [==========] - 4s 14ms/step - loss: 0.0516 - accuracy: 0.9871
```

Una vez entrenada y evaluada, predecimos. En este caso son 3.4GB de imágenes de prueba, que no adjuntamos junto a la práctica debido a su peso. En nuestro caso, hemos código una imagen de cada clase para hacer la predicción por imágenes, mostrando las 3 clases más posibles a la que pertenece cada imagen, que se muestra a continuación.

operating the radio 99.999985% // talking to passenger 1.4906156e-05% // safe driving 8.029095e-07%

talking on the phone - right 94.23743% // texting - right 4.9301076% // drinking 0.6259213%

hair and makeup 90.95711% // operating the radio 6.108872% // reaching behind 2.3821242%

hair and makeup 93.691086% // reaching behind 5.6924853% // drinking 0.3916315%

talking on the phone - left 46.290905% // drinking 35.720383% // texting - left 16.95104%

drinking 99.269936% // talking on the phone - right 0.39304462% // reaching behind 0.3305695%

hair and makeup 99.734985% // talking to passenger 0.09859587% // safe driving 0.06610907%

talking to passenger 99.94288% // hair and makeup 0.05551355% // operating the radio 0.00086506334%

hair and makeup 51.85162% // drinking 39.989037% // talking to passenger 7.549195%

Vemos como gran parte de las predicciones se cumplen o son cercanas a ellas. Además, hay que matizar que esto son solo 10 predicciones, lo cual es bastante difícil que acierte las 10. Viendo aún así, los valores de loss y accuracy, podemos saber que nuestra red está bien entrenada.

Por otro lado, hemos procesado las 80000 imágenes de prueba, creando un CSV que se adjunta en la entrega con las respectivas predicciones de cada imagen.

3.7 Augmentation

El augmentation es se traduce como el aumento de datos: una técnica para aumentar la diversidad de su conjunto de entrenamiento mediante la aplicación de transformaciones aleatorias (pero realistas), como la rotación de imágenes.

A las imágenes de prueba y de validación les aplicamos esta técnica aplicándole cambios. Estos cambios consisten en crear sintéticamente nuevos datos de entrenamiento aplicando algunas transformaciones en los datos de entrada. De esta manera a cada imagen le pasamos unos "filtros" que la distorsionan, de tal manera que entrenamos la red más a fondo puesto que no aprende a reconocer a una imagen con X características si no que amplia mucho su visión. En la siguiente imagen podemos ver un ejemplo de Augmentation:

Los resultados de la práctica anterior no se hicieron realizando esta técnica puesto que no se requería, pero sí que veíamos conveniente matizar el funcionamiento de este y como gracias a esta técnica nos puede quedar una red más completa que sea capaz de predecir imágenes de una forma más fiable y sin necesidad de un formato determinado. En el notebook de la práctica se adjunta la función de Augmentation y se aplica cambiando la llamada a la hora de leer los datos $train, validation = meterdatos(batch)$ por $train, validation = meterdatosaug()$. A continuación, se adjunta una foto del código que en este caso nos aplicaría los filtros expuestos en el $ImageDataGenerator$:

```
def meterdatosaug():

    train_datagen = ImageDataGenerator(  # Aqui se hacen los cambios a las imagenes
        rescale=1./255, # Normalizar
        shear_range=0.2, # Mover la imágen x pixeles
        zoom_range=0.2, # Zoom
        horizontal_flip=True, # 180º flip
        validation_split=0.2
    )

    test_datagen = ImageDataGenerator(rescale=1./255) # Simplemente normalizar

    train_generator = train_datagen.flow_from_directory(
        directory=r'../Dataset/imgs/train/',
        target_size=(xpixel, ypixel),
        batch_size=batch,
        class_mode='categorical')

    validation_generator = test_datagen.flow_from_directory(
        directory=r'../Dataset/imgs/train/',
        target_size=(xpixel, ypixel),
        batch_size=batch,
        class_mode='categorical')

    return train_generator, validation_generator
```

4 Práctica 2

4.1 Introducción

En esta práctica debemos crear otra red CNN, pero en este caso la red no debe identificar la acción del conductor, sino al conductor. Por lo que seguiremos los mismos pasos que en la práctica anterior, pero en este caso los valores con los que se comparara la salida serán adecuados para la identificación de dichos conductores.

Es por ello, que esta práctica es igual que la anterior, pero cambia las entradas. Por ello, la memoria de esta parte seguirá exactamente la misma estructura.

4.2 Estudio Redes

En el siguiente apartado realizaremos un estudio de redes neuronales en base a valores como el número de capas de convolución, el tamaño del MLP, el dropout…

El esquema que seguiremos para estudiar las redes será el siguiente:

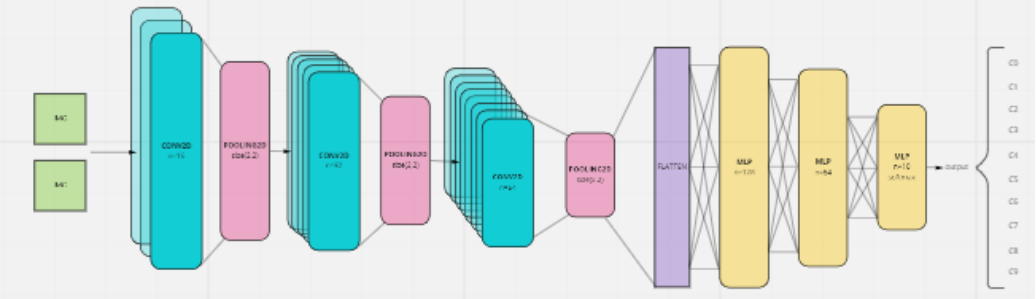
- En las cuatro primeras redes la influencia del número de capas de convolución y como afecta el dropout
- En las dos últimas redes la influencia del tamaño del MLP

4.2.1 Red 1

4.2.1.1 Descripción

En la primera red podemos ver la siguiente estructura. Tres capas de convolución con su respectivo pooling cada una. Por último, un MPL de 128 y 64 neuronas respectivamente.

Cuando añadimos el dropout, se añade en cada capa de convolución.



```
model.add(Conv2D(filters=n_conv1, kernel_size=(
    3, 3), padding='same', activation='relu'))
# Capas convolucionales
model.add(MaxPooling2D())
#   >este bloque se puede seguir añadiendo, quiza con menos neuronas,
# (3,3) es mucho se recomienda unsar 1x1 cuando las img no son mayores
model.add(Conv2D(n_conv2, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2)) # dropout

model.add(MaxPooling2D())
# model.add(Conv2D(n_conv3,(3,3),activation='relu',padding='same'))
model.add(Conv2D(n_conv3, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
model.add(Dropout(0.2))

# Capa fully-connected - MLP
model.add(Flatten())
model.add(Dense(128, activation='relu'))  # red fully-connected
model.add(Dense(64, activation='relu'))  # red fully-connected
```

<u>Sin DropOut (solo hacemos un periodo ya que los <= 0.2):</u>

`Loss : 0.0755 - accuracy : 0.9809 - val_loss : 4.2426e-04 - val_accuracy : 0.9998`

<u>Con DropOut:</u>

`Loss: 0.0900 - accuracy: 0.9787 - val_loss: 0.1820 - val_accuracy: 0.9886`

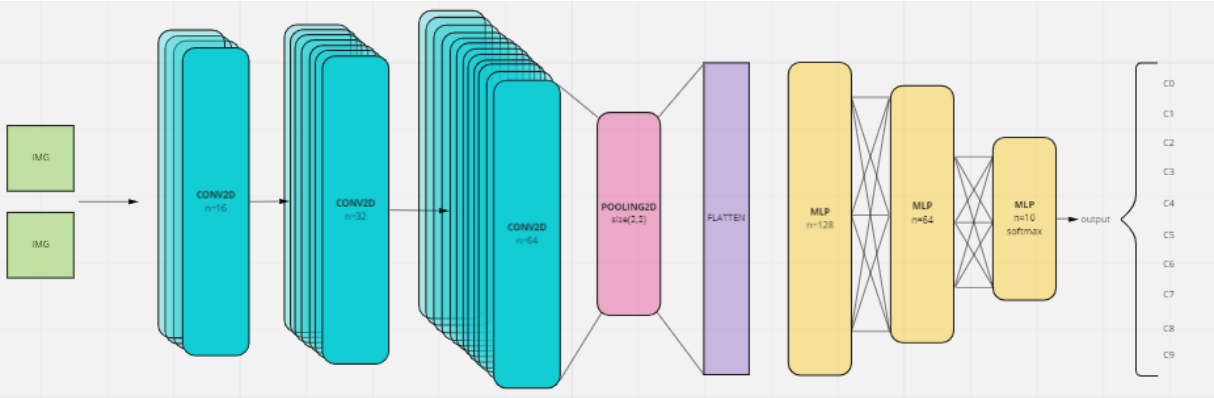
Podemos observar como solo en un epoch es capar de acertar el 99% de los datos de val, al ser val_accuracy mayor que el de entrenamiento nos cercioramos de que no hay overfitting, al añadir dropout reducimos como de rápido aprende además de emporar todos los parámetros en este caso.

4.2.2 Red 2

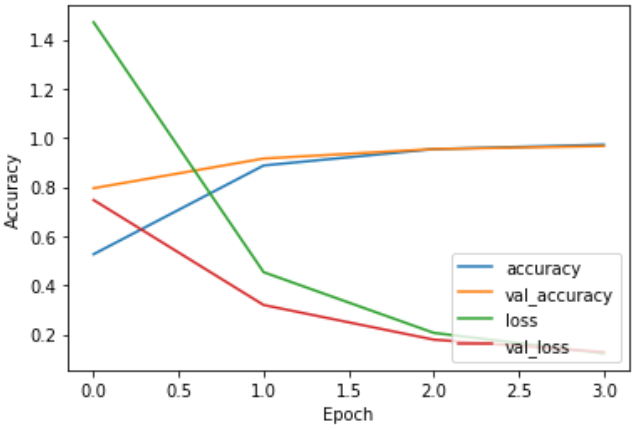
4.2.2.1 Descripción

En la segunda red se dan 3 capas de convolución seguidas con un único pooling al final. Por último, un MPL de 128 y 64 neuronas respectivamente.

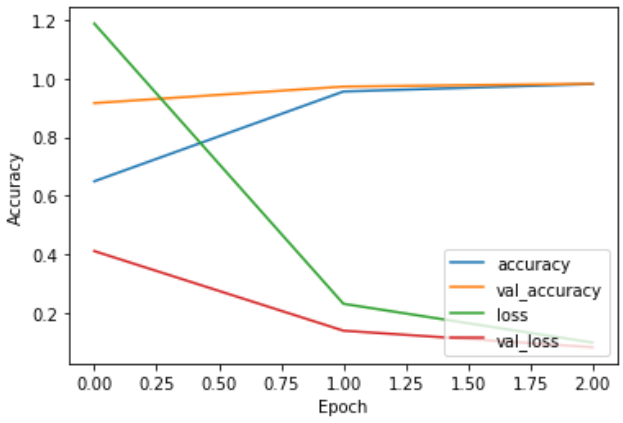
Cuando añadimos el dropout, se añade al final del pooling.




```
model.add(Conv2D(filters=n_conv1, kernel_size=(3, 3), padding='same', activation='relu'))
model.add(Conv2D(n_conv2, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(n_conv3, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D()) #model.add(MaxPooling2D(pool_size=(2,2)))
# model.add(Dropout(0.2))

# Capa fully-connected - MLP
model.add(Flatten())
model.add(Dense(128, activation='relu'))  # red fully-connected
model.add(Dense(64, activation='relu'))  # red fully-connected
```



Sin dropout



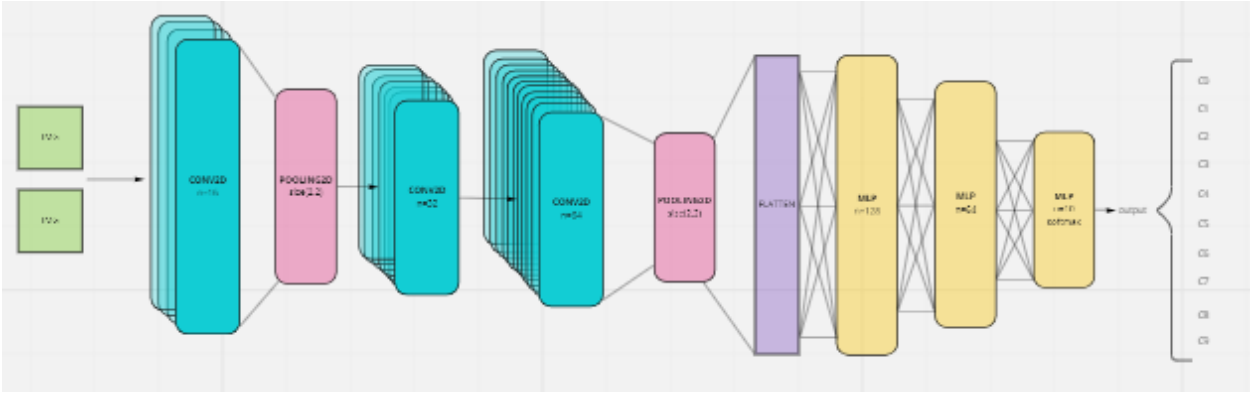
Con dropout

4.2.3 Red 3

4.2.3.1 Descripción

En la tercera red existen tres capas de convolución, con un pooling en la 1º y 3º capa de convolución. Por último, un MPL de 128 y 64 neuronas respectivamente.

Cuando añadimos el dropout, se añade al final del pooling.



```
model.add(Conv2D(filters=n_conv1, kernel_size=(3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D())
# capas convolucionales concatenadas
model.add(Conv2D(n_conv2, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(n_conv3, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D()) #model.add(MaxPooling2D(pool_size=(2,2)))
# model.add(Dropout(0.2))

# Capa fully-connected - MLP
model.add(Flatten())
model.add(Dense(128, activation='relu'))  # red fully-connected
model.add(Dense(64, activation='relu'))  # red fully-connected
```

<u>Sin DropOut (solo hacemos un periodo ya que los <= 0.2):</u>

`loss: 0.0890 - accuracy: 0.9802 - val_loss: 0.0013 - val_accuracy: 0.9998`

<u>Con DropOut:</u>

`loss: 0.0845 - accuracy: 0.9824 - val_loss: 0.0071 - val_accuracy: 0.9989`

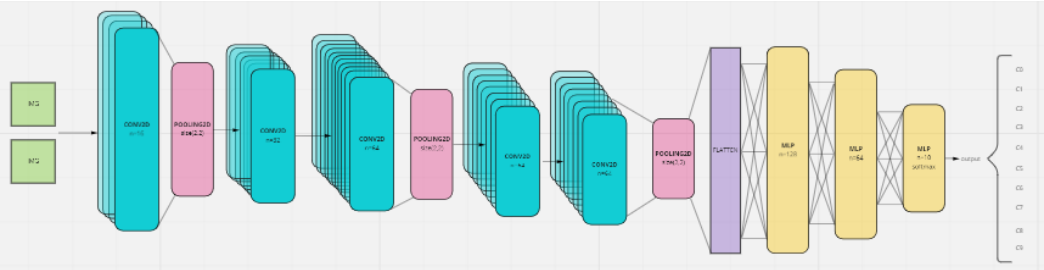
Aquí vemos como al aumentar la última capa nos quedamos con un val_loss mayor que en el de la red 1 siendo los demás parámetros iguales, lo que implica que tampoco hay overfitting (acurracy < val_accuracy) pero aun podemos mejorar en comparación con la red 1 que en el mismo epoch tenía un loss mucho menor.

4.2.4 Red 4

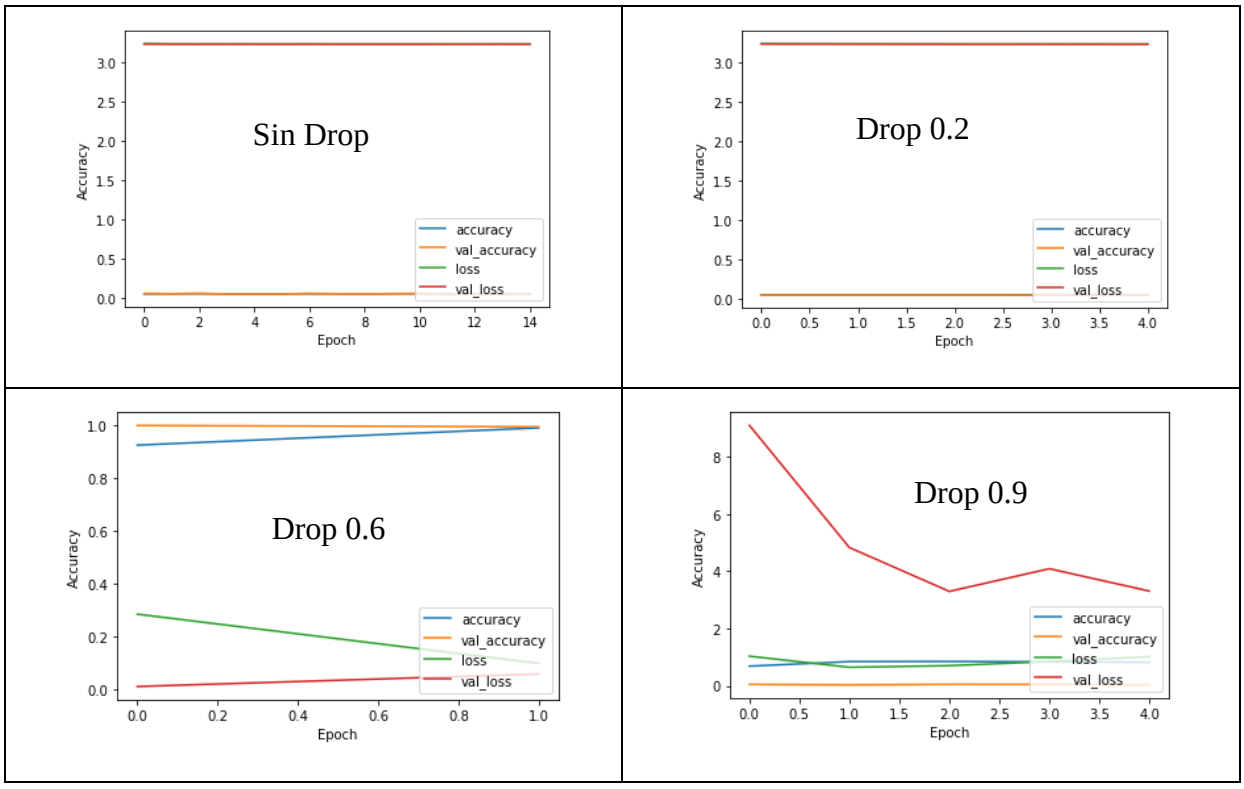
4.2.4.1 Descripción

En la cuarta red existen cinco capas de convolución, con un pooling en la 1º, 3º y 5º capa de convolución. Por último, un MPL de 128 y 64 neuronas respectivamente.

Cuando añadimos el dropout, se añade al final del pooling.



```
model.add(Conv2D(filters=n_conv1, kernel_size=(3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D())
model.add(Conv2D(n_conv2, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(n_conv3, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D()) #model.add(MaxPooling2D(pool_size=(2,2)))
# model.add(Dropout(0.2))
model.add(Conv2D(n_conv3*2, (3, 3), activation='relu', padding='same'))    # lo multiplico
model.add(Conv2D(n_conv3*4, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
# Capa fully-connected - MLP
model.add(Flatten())
model.add(Dense(128, activation='relu'))  # red fully-connected
model.add(Dense(64, activation='relu'))  # red fully-connected
```



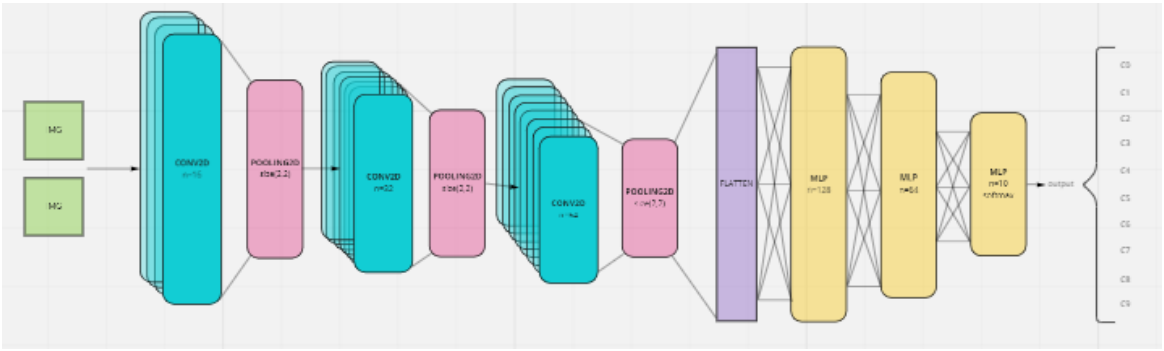
Al observa las 4 graficas de la red 4 nos damos cuenta de que no aprende ya que nuestra red es demasiado profunda y no es capaz de entrenar todos los parámetros, es por ello que al aumentar el dropout vemos una mejora notable en el loss y otra sutil en los accuracy ya que se reduce el número de parámetros.

4.2.5 Red 5

4.2.5.1 Descripción

En la primera red podemos ver la siguiente estructura. Tres capas de convolución con su respectivo pooling cada una. Por último, un MPL de 32 y 16 neuronas respectivamente.

Cuando añadimos el dropout, se añade en cada capa de convolución.



```
model.add(Conv2D(filters=n_conv1, kernel_size=(3, 3), padding='same', activation='relu'))
# Capas convolucionales
model.add(MaxPooling2D())
#   >este bloque se puede seguir añadiendo, quiza con menos neuronas, o menos capas convolu
# (3,3) es mucho se recomienda unsar 1x1 cuando las img no son mayores de 128x128 la nuestr
model.add(Conv2D(n_conv2, (3, 3), activation='relu', padding='same'))
# model.add(Dropout(0.2)) # dropout

model.add(MaxPooling2D())
# model.add(Conv2D(n_conv3,(3,3),activation='relu',padding='same'))
model.add(Conv2D(n_conv3, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
# model.add(Dropout(0.2))

# Capa fully-connected - MLP
model.add(Flatten())
model.add(Dense(32, activation='relu'))  # red fully-connected
model.add(Dense(32/2, activation='relu'))  # red fully-connected
```

<u>Sin DropOut (solo hacemos un periodo ya que los <= 0.2):</u>

`loss: 0.1269 - accuracy: 0.9634 - val_loss: 0.0014 - val_accuracy: 0.9993`

Aquí podemos ver una mejora notable en la generalización de esta red, es verdad que el accuracy de entrenamiento ha bajado respecto a otras redes, pero hemos mejorado el val_accuracy

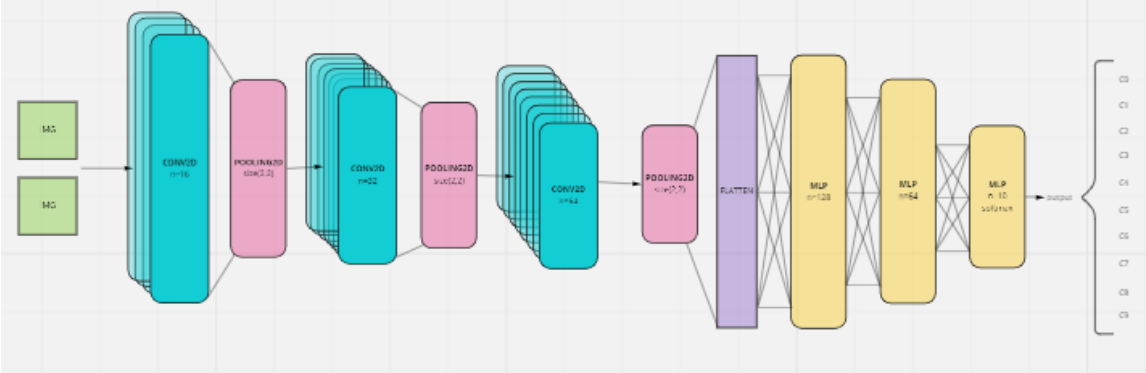
lo que implica que generaliza más que las redes anteriores la reducción del accuracy se debe a que no memoriza tanto como las otras redes, provocando que falle el ruido que antes memorizaba.

4.2.6 Red 6

4.2.6.1 Descripción

En la primera red podemos ver la siguiente estructura. Tres capas de convolución con su respectivo pooling cada una. Por último, un MPL de 1024 y 512 neuronas respectivamente.

Cuando añadimos el dropout, se añade en cada capa de convolución.



```
model.add(Conv2D(filters=n_conv1, kernel_size=(3, 3), padding='same', activation='relu'))
# Capas convolucionales
model.add(MaxPooling2D())
#   >este bloque se puede seguir añadiendo, quiza con menos neuronas, o menos capas convol
# (3,3) es mucho se recomienda unsar 1x1 cuando las img no son mayores de 128x128 la nuest
model.add(Conv2D(n_conv2, (3, 3), activation='relu', padding='same'))
# model.add(Dropout(0.2)) # dropout

model.add(MaxPooling2D())
# model.add(Conv2D(n_conv3,(3,3),activation='relu',padding='same'))
model.add(Conv2D(n_conv3, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
# model.add(Dropout(0.2))

# Capa fully-connected - MLP
model.add(Flatten())
model.add(Dense(1024, activation='relu'))  # red fully-connected
model.add(Dense(1024/2, activation='relu'))  # red fully-connected
```

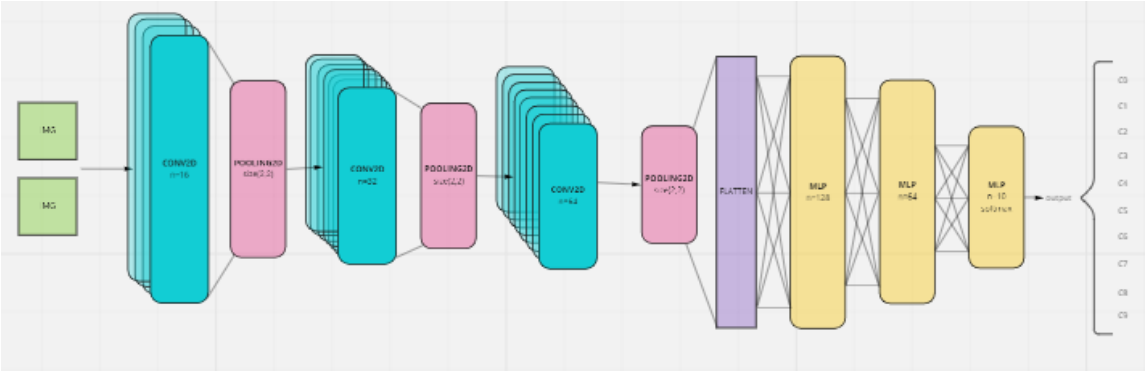
<u>Sin DropOut (solo hacemos un periodo ya que los <= 0.2):</u>

`loss: 0.1096 - accuracy: 0.9763 - val_loss: 4.8323e-06 - val_accuracy: 1.0000`

Esta es una mejora de la red 5, lo que sacamos en claro es que necesitábamos más neuronas en nuestro mlp.

4.3 Conclusión de la mejor red

Como hemos podido observar la mejor red dado los datos obtenidos es la red 5 ya que no memoriza como podemos ver en su accuracy y generaliza muy bien ya que tiene un 100% de precisión con los datos de validación.



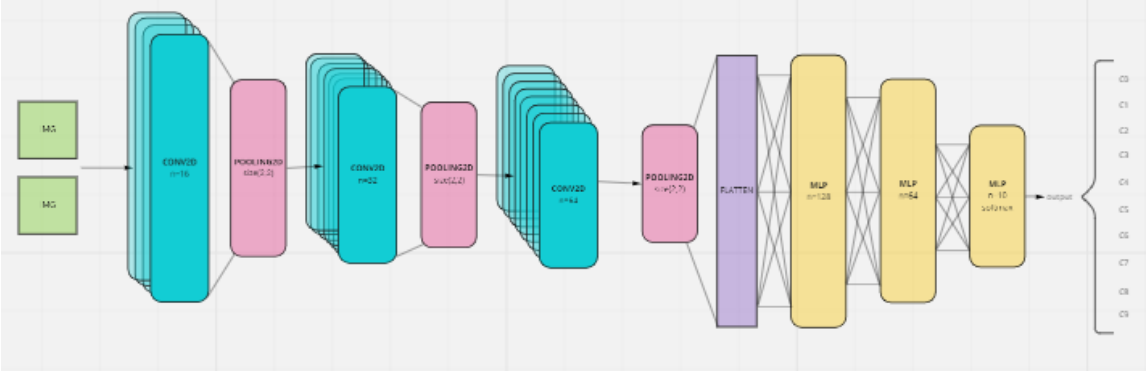

```
model.add(Conv2D(filters=n_conv1, kernel_size=(3, 3), padding='same', activation='relu'))
# Capas convolucionales
model.add(MaxPooling2D())
#   >este bloque se puede seguir añadiendo, quiza con menos neuronas, o menos capas convol
# (3,3) es mucho se recomienda unsar 1x1 cuando las img no son mayores de 128x128 la nuest
model.add(Conv2D(n_conv2, (3, 3), activation='relu', padding='same'))
# model.add(Dropout(0.2)) # dropout

model.add(MaxPooling2D())
# model.add(Conv2D(n_conv3,(3,3),activation='relu',padding='same'))
model.add(Conv2D(n_conv3, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
# model.add(Dropout(0.2))

# Capa fully-connected - MLP
model.add(Flatten())
model.add(Dense(1024, activation='relu'))  # red fully-connected
model.add(Dense(1024/2, activation='relu'))  # red fully-connected
```

loss: 0.1096 - accuracy: 0.9763 - val_loss: 4.8323e-06 - val_accuracy: 1.0000

4.4 Estudio del learning rate

En esta práctica como en la anterior hemos podido ver que el learning rate de 0.001 es el mejor dado el problema a resolver, hemos aplicado el mismo estudio que en la parte anterior

4.5 Pruebas de red



Sabiendo esto, queda la parte final, la de clasificar el conjunto de imágenes de prueba, recogiendo en la memoria la salida obtenida, además de un CSV para todas las imágenes de prueba.

Es por ello por lo que primero entrenamos nuestra red, que como resultado nos da los siguientes valores de accuracy en un epoch:

```
loss: 0.0847 - accuracy: 0.9776 - val_loss: 1.6476e-04 - val_accuracy: 1.0000
```

Además, también realizamos la evaluación de nuestra red con los datos de entrenamiento y de validación, sacándonos el siguiente resultado:

```
1122/1122 [==============================] - 69s 61ms/step - loss: 3.1175e-04 - accuracy: 0.9999
281/281 [==============================] - 17s 60ms/step - loss: 1.6476e-04 - accuracy: 1.0000
```


p016 100.0% // p015 2.8040859e-07% // p014 2.7065294e-08%
p012 100.0% // p024 2.308757e-07% // p081 1.7690579e-07%
p039 100.0% // p056 2.738979e-06% // p047 1.8071429e-06%
p052 99.999985% // p056 6.469726e-06% // p051 2.3640708e-07%
p015 99.99996% // p014 3.1304957e-05% // p012 2.1086166e-06%
p047 99.999985% // p039 1.5746908e-05% // p045 6.715844e-07%

Conclusión

Una vez concluidas las dos practicas hemos logrado comprender y asimilar la teoría vista en clase, asociando los conocimientos a problemas de la vida real utilizando métodos de inteligencia artificial. Esta práctica nos ha hecho enfrentarnos a posibles problemas de la vida real, como podría ser una cámara de tráfico encargada de predecir al conductor. Además, hemos podido aprender metodologías tremendamente importantes como la de Augmentation o la de lectura de ficheros por su clase.