Visión por Computador

Práctica 2:
Redes Neuronales Convolucionales

Curso 2019-2020
Cuarto Curso del Grado en Ingeniería Informática

Laura Rabadán Ortega 79088745W
laurarabadan@correo.ugr.es

Contenido

1. BASENET EN CIFAR100

Para este primer apartado, se pedía en la práctica que se reprodujera con Keras el modelo llamado BaseNet:

Layer No.	Layer Type	Kernel size (for conv layers)	Input | Output dimension	Input | Output channels (for conv layers)
1	Conv2D	5	32 | 28	3 | 6
2	Relu	-	28 | 28	-
3	MaxPooling2D	2	28 | 14	-
4	Conv2D	5	14 | 10	6 | 16
5	Relu	-	10 | 10	-
6	MaxPooling2D	2	10 | 5	-
7	Linear	-	400 | 50	-
8	Relu	-	50 | 50	-
9	Linear	-	50 | 25	-

Esta arquitectura se traduce a código, utilizando las siguientes capas:

- Conv2D: capa que realiza convoluciones 2D. Recibe como parámetros el número de canales que se consigue después de esta capa, la dimensión del kernel de convolución, la función de activación que se va a utilizar y, en el caso de que sea la primera capa del modelo, las dimensiones que tienen las imágenes de entrada.
- Maxpooling2D: reduce el número de datos quedándose con el valor más alto de cada región de tamaño determinado, especificado como parámetro de entrada.
- Flatten: se encarga de alinear un bloque de características, convertirlo en un vector.
- Dense: capa totalmente conectada que recibe como parámetros de entrada el número de canales que se conseguirán de salida y la función de activación que se quiere utilizar.
- Activation: se añade una función de activación.

Por tanto, basándonos en la tabla anterior y utilizando las funciones descritas, se consigue el siguiente código:

```
model = Sequential()

model.add(Conv2D(6,(5,5),activation='relu',input_shape=(32,32,3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(16,(5,5),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten()

model.add(Dense(50,activation='linear'))
model.add(Activation('relu'))
model.add(Dense(25, activation='linear'))
model.add(Activation('softmax'))
```

El modelo anterior se compila y se entrena con el conjunto de *train*, indicándole la función de pérdida, el optimizador y la métrica que se va a utilizar (compilador); y el conjunto de imágenes que se va a utilizar para entrenar la red junto con sus etiquetas, el tamaño de *batch* que se quiere utilizar, el número de épocas que se van a realizar y el conjunto de validación.

```
model.compile(loss=keras.losses.categorical_crossentropy,optimizer=SGD(),
              metrics=['accuracy'])

model.fit(x_train,y_train,batch_size=32,epochs=15,verbose=1,
          validation_split=0.1)
```

Si se ejecuta todo lo anterior, se consiguen los siguientes resultados:

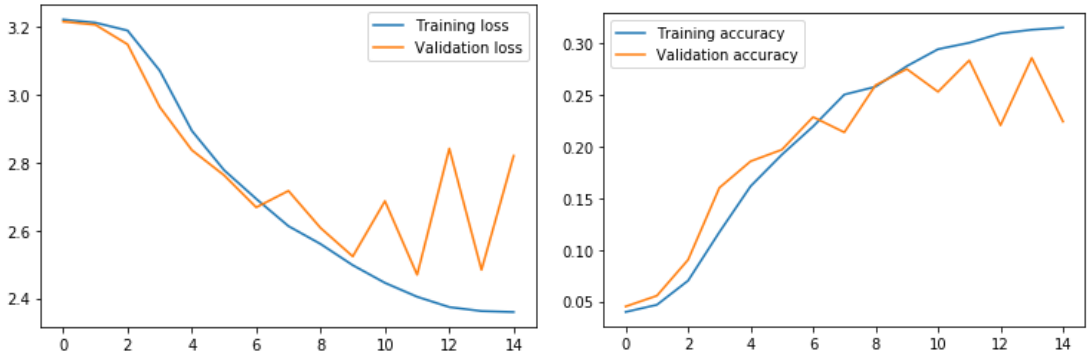
- Resumen de la arquitectura del modelo definido:

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 28, 28, 6)         456
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 10, 10, 16)        2416
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 5, 5, 16)          0
_________________________________________________________________
flatten_7 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_21 (Dense)             (None, 50)                20050
_________________________________________________________________
activation_364 (Activation)  (None, 50)                0
_________________________________________________________________
dense_22 (Dense)             (None, 25)                1275
_________________________________________________________________
activation_365 (Activation)  (None, 25)                0
=================================================================
Total params: 24,197
Trainable params: 24,197
Non-trainable params: 0
_________________________________________________________________
```

- Proceso de entrenamiento:

```
Train on 10000 samples, validate on 2500 samples
Epoch 1/15
10000/10000 [==============================] - 22s 2ms/step - loss: 3.1806 - acc: 0.0728 - val_loss: 3.1179 - val_acc: 0.1184
Epoch 2/15
10000/10000 [==============================] - 12s 1ms/step - loss: 3.0086 - acc: 0.1345 - val_loss: 2.8972 - val_acc: 0.1564
Epoch 3/15
10000/10000 [==============================] - 12s 1ms/step - loss: 2.8360 - acc: 0.1840 - val_loss: 2.7336 - val_acc: 0.2024
Epoch 4/15
10000/10000 [==============================] - 11s 1ms/step - loss: 2.6820 - acc: 0.2218 - val_loss: 2.5830 - val_acc: 0.2472
Epoch 5/15
10000/10000 [==============================] - 12s 1ms/step - loss: 2.5671 - acc: 0.2525 - val_loss: 2.6399 - val_acc: 0.2404
Epoch 6/15
10000/10000 [==============================] - 12s 1ms/step - loss: 2.4921 - acc: 0.2755 - val_loss: 2.5024 - val_acc: 0.2676
Epoch 7/15
10000/10000 [==============================] - 11s 1ms/step - loss: 2.4294 - acc: 0.2942 - val_loss: 2.5489 - val_acc: 0.2476
Epoch 8/15
10000/10000 [==============================] - 12s 1ms/step - loss: 2.3769 - acc: 0.3045 - val_loss: 2.5621 - val_acc: 0.2508
Epoch 9/15
10000/10000 [==============================] - 12s 1ms/step - loss: 2.3527 - acc: 0.3152 - val_loss: 2.5383 - val_acc: 0.2700
Epoch 10/15
10000/10000 [==============================] - 11s 1ms/step - loss: 2.3396 - acc: 0.3194 - val_loss: 2.9909 - val_acc: 0.1872
Epoch 11/15
10000/10000 [==============================] - 11s 1ms/step - loss: 2.3340 - acc: 0.3214 - val_loss: 2.5063 - val_acc: 0.2920
Epoch 12/15
10000/10000 [==============================] - 11s 1ms/step - loss: 2.3176 - acc: 0.3298 - val_loss: 2.6244 - val_acc: 0.2664
Epoch 13/15
10000/10000 [==============================] - 11s 1ms/step - loss: 7.5689 - acc: 0.2048 - val_loss: 15.3831 - val_acc: 0.0456
Epoch 14/15
10000/10000 [==============================] - 11s 1ms/step - loss: 15.4959 - acc: 0.0386 - val_loss: 15.3831 - val_acc: 0.0456
Epoch 15/15
10000/10000 [==============================] - 11s 1ms/step - loss: 15.4959 - acc: 0.0386 - val_loss: 15.3831 - val_acc: 0.0456
```

- Porcentaje de *accuracy* conseguido: `22.68 %`
- Gráficas de la evolución de los valores de la función de pérdida y del *accuracy* conseguidos durante el entrenamiento del modelo:



2. MEJORA DEL MODELO

En este apartado, basándonos en el modelo anterior, se van a realizar distintos experimentos para, en cada punto, encontrar los mejores parámetros e intentar conseguir un modelo mejor que el inicial, fruto de la combinación de las modificaciones que dieron mejor resultado cuando se probaron.

2.1. Normalización de datos

En este primer caso, no se realiza ningún tipo de *experimento*, simplemente se añade la normalización a los datos del conjunto de *train* y a los de *test* y se compara el resultado obtenido con el modelo original, para comprobar el porcentaje de mejora que se consigue.

Para poder normalizar los datos de los conjuntos de *train* y *test*, se utilizan las siguientes sentencias:

```
ImageDataGenerator(featurewise_center=True,
                   featurewise_std_normalization=True)
datagen.fit(x)
```

Tanto para el conjunto de entrenamiento como para el de test. Al utilizar *ImageDataGenerator* para modificar los datos de entrada, se tiene modificar la línea de entrenamiento del modelo a:

```
model.fit_generator(datagen_train.flow(x_train,y_train,batch_size=32,
                    subset='training'),verbose=1,epochs=15,
                    validation_data=datagen_train.flow(x_train,y_train,
                    batch_size=32,subset='validation'))
```

Ejecutando este modelo y comparándolo con una ejecución del modelo original, se obtiene que, solo con normalizar los datos de entrada, se consigue una mejora del 25% con respecto al modelo original.

Modelo	***Accuracy***
Básico	31.68 %
Normalización	39.88 %
Porcentaje de mejora	25.88 %

2.2. Aumento de datos

Para el aumento de datos se van a probar distintos parámetros, que son:

- *Zoom_range*: hace un zoom aleatorio dentro del rango establecido, con el parámetro pasado como argumento. Se prueba con los valores 0.2 y 1.2.
- *Rotation_range*: rota las imágenes aleatoriamente en función a un grado especificado. Se prueba con los valores 20 y 90.
- *Horizontal_flip*: de manera aleatoria, gira horizontalmente imágenes de entrada.

Estos parámetros se añaden al *ImageDataGenerator* del conjunto de entrenamiento. Por tanto, primero se ejecuta el modelo con cada parámetro de manera independiente. Después, se combina *zoom_range* y *rotation_range* con *horizontal_flip* y por último se ejecutan los 3 parámetros juntos. Los resultados que se consiguen son los siguientes:

Modelo	*Accuracy*
Normalización	39.88 %
Zoom 0.2	41.48 %
Zoom 1.2	33.60 %
Rotation 20	39.60 %
Rotation 90	35.00 %
Horizontal flip	41.48 %
Zoom 0.2 + flip	41.32 %
Rotation 20 + flip	40.96 %
Rotation 90 + flip	35.04 %
Zoom 0.2 + rotation 20 + flip	39.24 %

Se decide probar a combinar los parámetros solo con *horizontal_flip* para intentar mejorar los valores, ya que con esa modificación se consigue uno de los mejores resultados.

En vista de los resultados, se decide escoger para modificar los datos *horizontal_flip* con *zoom_range* de 0.2, ya que son los que mejores valores consiguen por separado y, de las combinaciones, la mejor de todas. Con esto se consigue una mejora del 30% con respecto al modelo original, y del 5% con respecto a la mejora anterior.

Modelo	*Accuracy*
Básico	31.68 %
Aumento de datos	41.32 %
Porcentaje de mejora	30.43 %

Modelo	*Accuracy*
Normalización	39.88 %
Aumento de datos	41.32 %
Porcentaje de mejora	5.02 %

2.3. Red más profunda

Al igual que en el apartado anterior, se proponen distintos modelos para buscar el que consigue mejores resultados. Se prueba:

- Añadir una capa Conv2D de 3x3 después de cada capa original Conv2D de 5x5.
- Añadir una tercera capa Conv2D de 5x5 y modificar los Maxpooling, añadiendo y quitando capas de ese tipo.
- Añadir tres capas Conv2D de 3x3 después de cada capa original Conv2D de 5x5.

Con estos ejemplos se quiere comprobar la manera en la que afectan las capas de maxpooling a los resultados finales y si es preferible tener muchas capas de convoluciones pequeñas o menos capas con convoluciones mayores.

Una vez se consigue saber que modelo es mejor, se prueba a modificar el número de canales de salida de cada capa, para ver si influye, en que manera y que valores son los mejores.

Por tanto, en primer, se prueba que modelo es preferible de manera general. Se ejecuta cada una de las ideas expuestas y se consiguen los siguientes resultados:

Modelo	***Accuracy***
Aumento de datos	41.32 %
Conv2D 3x3 (2, antes de los maxpooling)	37.56 %
Conv2D 3x3 (2, después de los maxpooling)	37.00 %
Conv2D 5x5 + 1º sin Maxpooling	41.52 %
Conv2D 5x5 + 2º sin Maxpooling	36.08 %
Conv2D 5x5 + 1º/2º sin Maxpooling	44.96 %
Conv2D 3x3 (5 – 3 – 3 – 3 – 5 – 3 – 3 – 3)	28.80 %
Conv2D 3x3 (5 – 3 – 3 – 3 – 3 – 3 – 3 – 5)	33.28 %

El mejor resultado que se consigue es añadiendo una capa más de Conv2D de 5x5 y tener una única capa de MaxPooling al final de la ultima capa de Conv2D. A raíz de esa arquitectura, se prueban distintos tamaños de salida de los canales de las capas de Conv2D y la mejor configuración que se consigue, con un 49.48% de *accuracy*, es la siguiente:

Layer No.	Layer Type	Kernel size (for conv layers)	Input | Output dimension	Input | Ouput channels (for conv layers)
1	Conv2D	5	32 | 28	3 | 6
2	Relu	-	28 | 28	-
3	Conv2D	5	28 | 24	6 | 56
4	Relu	-	24 | 24	-
5	Conv2D	5	24 | 20	6 | 106
6	Relu	-	20 | 20	-
7	MaxPooling	2	20 | 10	-
8	Flatten	-	10 | 400	-
9	Linear	-	400 | 50	-
10	Relu	-	50 | 50	-
11	Linear	-	50 | 25	-
12	SoftMax	-	25 | 25	

Lo que demuestra que la cantidad de canales de salida de las capas Conv2D es importante para lograr mejores resultados.

Con esta modificación se consigue una mejora del 56% con respecto al modelo original, y del 19% con respecto a la modificación anterior.

Modelo	***Accuracy***
Básico	31.68 %
Red más profunda	49.48 %
Porcentaje de mejora	56.18 %

Modelo	***Accuracy***
Aumento de datos	41.32 %
Red más profunda	49.48 %
Porcentaje de mejora	19.75 %

2.4. Capas de normalización

En este apartado se pide añadir a la arquitectura que se tiene capas de normalización, *BatchNormalization*, y se pruebe a ponerla antes y después de las capas ReLU. Se ejecutan distintas configuraciones para las posiciones de las capas de normalización, y los resultados que se consiguen son los siguientes:

Modelo	*Accuracy*
Red más profunda	49.48 %
Después de ReLU (todas)	54.48 %
Antes de ReLU (todas)	53.80 %
Después ReLU (Conv2D) + Antes ReLU (Dense)	52.84 %
Antes ReLU (Conv2D) + Después ReLU (Dense)	51.44 %

Tras lo resultados de las experimentaciones, se ve que es mejor añadir una capa de normalización después de la capa de activación ReLU. Este es el modelo que se selecciona, con el que se consigue obtener un 72% de mejora con respecto al modelo original y un 10% de mejora respecto al que se tenía hasta el momento.

Modelo	*Accuracy*
Básico	31.68 %
Capas normalización	54.48 %
Porcentaje de mejora	72.00 %

Modelo	*Accuracy*
Red más profunda	49.48 %
Capas normalización	54.48 %
Porcentaje de mejora	10.11 %

2.5. *"Early Stopping"*

Por último, se va a estudiar el número de épocas que se necesitan para conseguir buenos resultados. Para probar esto, se van a realizar distintas ejecuciones con distinto número de épocas (5, 10, 15, 20, 25, 30) y una ejecución con *Early Stopping* con máximo de épocas a 50 para poder comparar los resultados y seleccionar el número de épocas con el que se entrenará nuestro modelo finalmente.

Modelo	*Accuracy*
Capas de normalización	54.48 %
5 ejecuciones	46.92 %
10 ejecuciones	52.08 %
15 ejecuciones	53.80 %
20 ejecuciones	55.08 %
25 ejecuciones	55.52 %
30 ejecuciones	56.60 %
Early Stopping (50 ejecuciones – 11ª ejecución)	52.52 %

El peor resultado, cómo era de esperar, es en el que se realizan 5 ejecuciones, ya que a la red no le da tiempo a entrenar nada, por lo que no va a poder clasificar de manera correcta. El mejor resultado es tras 30 ejecuciones, en cambio, se selecciona el de 20. Se descartan las 25 y 30 épocas porque, aunque consiguen mejores resultados, están muy pegados a los datos, la red se empieza a sobreentrenar. No se puede tener una alta seguridad que en todo momento vaya a

conseguir en test un valor tan alto, por lo que es preferible escoger un número de épocas menor a costa de conseguir un *accuracy* menor pero seguro.

Al ejecutar el *EarlyStooping*, muestra que en la época 11 podría dejarse de entrenar el modelo. Podría ser una buena opción también, pero se opta por las 20 épocas porque se consiguen mejores resultados y no parece ser un número muy alto como para que la red se sobre entrene, y se consigue explicar casi perfectamente casi perfectamente los catos del *train*.

Por tanto, comparando con el modelo original se consigue un 74% de mejora, un 1% de mejora con respecto a la capa anterior.

Modelo	***Accuracy***
Básico	31.68 %
Early Stooping	55.08 %
Porcentaje de mejora	73.86 %

Modelo	***Accuracy***
Capa normalización	54.48 %
Early Stooping	55.08 %
Porcentaje de mejora	1.10 %

2.6. Resumen

Tras seleccionar los parámetros que se van a escoger para cada sección se consigue un modelo que logra un valor de *accuracy* mejor que el del modelo inicial en un 74%. Para este modelo se hace un preprocesamiento de los datos, normalizándolos y aumentando el número de imágenes, con un *zoom_range* de 0.2 y una rotación horizontal aleatoria (solo al conjunto train) y se entrena en la siguiente arquitectura:

Layer No.	Layer Type	Kernel size (for conv layers)	Input | Output dimension	Input | Ouput channels (for conv layers)
1	Conv2D	5	32 | 28	3 | 6
2	Relu	-	28 | 28	-
3	BatchNormalization	-	28 | 28	-
4	Conv2D	5	28 | 24	6 | 56
5	Relu	-	24 | 24	-
6	BatchNormalization	-	24 | 24	-
7	Conv2D	5	24 | 20	56 | 106
8	Relu	-	20 | 20	-
9	BatchNormalization	-	20 | 20	-
10	MaxPooling	2	20 | 10	-
11	Flatten	-	10 | 400	-
12	Linear	-	400 | 50	-
13	Relu	-	50 | 50	-
14	BatchNormalization	-	50 | 50	-
15	Linear	-	50 | 25	-
16	BatchNormalization	-	25 | 25	-
17	SoftMax	-	25 | 25	-

Tras ejecutar este modelo, entrenado con 20 épocas y un conjunto de validación del 20% del *train*, se consiguen los siguientes datos:

- Resumen de la arquitectura del modelo definido:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_48 (Conv2D)           (None, 28, 28, 6)         456
_________________________________________________________________
batch_normalization_80 (Batc (None, 28, 28, 6)         24
_________________________________________________________________
conv2d_49 (Conv2D)           (None, 24, 24, 56)        8456
_________________________________________________________________
batch_normalization_81 (Batc (None, 24, 24, 56)        224
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 20, 20, 106)       148506
_________________________________________________________________
batch_normalization_82 (Batc (None, 20, 20, 106)       424
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 10, 10, 106)       0
_________________________________________________________________
flatten_17 (Flatten)         (None, 10600)             0
_________________________________________________________________
dense_37 (Dense)             (None, 50)                530050
_________________________________________________________________
activation_144 (Activation)  (None, 50)                0
_________________________________________________________________
batch_normalization_83 (Batc (None, 50)                200
_________________________________________________________________
dense_38 (Dense)             (None, 25)                1275
_________________________________________________________________
batch_normalization_84 (Batc (None, 25)                100
_________________________________________________________________
activation_145 (Activation)  (None, 25)                0
=================================================================
Total params: 689,715
Trainable params: 689,229
Non-trainable params: 486
```

- Proceso de entrenamiento:

```
Epoch 4/20
313/313 [==============================] - 77s 246ms/step - loss: 2.0260 - acc: 0.4313 - val_loss: 2.0457 - val_acc: 0.4212
Epoch 5/20
313/313 [==============================] - 77s 245ms/step - loss: 1.9172 - acc: 0.4625 - val_loss: 1.9562 - val_acc: 0.4456
Epoch 6/20
313/313 [==============================] - 77s 246ms/step - loss: 1.8459 - acc: 0.4794 - val_loss: 1.9117 - val_acc: 0.4512
Epoch 7/20
313/313 [==============================] - 77s 245ms/step - loss: 1.7610 - acc: 0.5038 - val_loss: 1.8401 - val_acc: 0.4756
Epoch 8/20
313/313 [==============================] - 77s 245ms/step - loss: 1.6785 - acc: 0.5207 - val_loss: 1.7721 - val_acc: 0.4924
Epoch 9/20
313/313 [==============================] - 77s 246ms/step - loss: 1.6252 - acc: 0.5365 - val_loss: 1.7976 - val_acc: 0.4952
Epoch 10/20
313/313 [==============================] - 77s 246ms/step - loss: 1.5626 - acc: 0.5517 - val_loss: 1.8191 - val_acc: 0.4784
Epoch 11/20
313/313 [==============================] - 77s 245ms/step - loss: 1.5099 - acc: 0.5661 - val_loss: 1.6873 - val_acc: 0.5124
Epoch 12/20
313/313 [==============================] - 77s 245ms/step - loss: 1.4574 - acc: 0.5849 - val_loss: 1.6854 - val_acc: 0.5096
Epoch 13/20
313/313 [==============================] - 77s 245ms/step - loss: 1.4247 - acc: 0.5878 - val_loss: 1.7000 - val_acc: 0.5084
Epoch 14/20
313/313 [==============================] - 77s 245ms/step - loss: 1.3719 - acc: 0.6080 - val_loss: 1.6984 - val_acc: 0.5088
Epoch 15/20
313/313 [==============================] - 77s 246ms/step - loss: 1.3201 - acc: 0.6178 - val_loss: 1.6219 - val_acc: 0.5284
Epoch 16/20
313/313 [==============================] - 77s 245ms/step - loss: 1.2899 - acc: 0.6258 - val_loss: 1.6519 - val_acc: 0.5184
Epoch 17/20
313/313 [==============================] - 77s 245ms/step - loss: 1.2427 - acc: 0.6417 - val_loss: 1.6965 - val_acc: 0.5100
Epoch 18/20
313/313 [==============================] - 77s 245ms/step - loss: 1.2056 - acc: 0.6511 - val_loss: 1.6006 - val_acc: 0.5340
Epoch 19/20
313/313 [==============================] - 76s 243ms/step - loss: 1.1766 - acc: 0.6595 - val_loss: 1.6609 - val_acc: 0.5108
Epoch 20/20
313/313 [==============================] - 77s 244ms/step - loss: 1.1434 - acc: 0.6680 - val_loss: 1.6862 - val_acc: 0.5104
```

- Porcentaje de *accuracy* conseguido: `55.6 %`
- Gráficas de la evolución de los valores de la función de pérdida y del *accuracy* conseguidos durante el entrenamiento del modelo:

2.7. Conclusiones

Con este estudio que se ha hecho para conseguir una red dada, se destaca la importancia del preprocesamiento de los datos antes de entrenar la red y, en especial, de la normalización de los datos. Simplemente normalizando el conjunto de datos, se consigue un 25% de mejora. Otra de las mejoras que destaca por su porcentaje de mejora, aproximadamente de un 20%, es el hacer la red más profunda, añadiendo más capas al modelo. Al añadir más capas se consigue extraer mayor número de características y, en consecuencia, más información para una mejor clasificación de nuevas imágenes.

3. TRANSFERENCIA DE MODELOS Y AJUSTE FINO CON RESNET50 PARA LA BASE DE DATOS CALTECH-UCSD

Este último apartado se divide en dos partes: utilizar la red ResNet50 como extractor de características y reentrenar la red para con nuestro conjunto de datos.

Para poder ejecutar la práctica se utiliza *colab*, descargando las imágenes en el servidor desde la carpeta de drive, en vez de ir leyéndolas una a una.

3.1. ResNet50 como extractor de características

Primero se va a montar un modelo que consiste en dos partes: ResNet50 entrenada con *Imagenet* y sin la última capa, y una pequeña red totalmente conectada que será la que se encargue de clasificar las imágenes. Para ello, lo primer que se hace es procesar los datos de los dos conjuntos, *train* y *test*, para pasárselos a ResNet50:

```
# Preprocesamiento de los conjuntos de datos de train y test
x_train = preprocess_input(x_train)
x_test = preprocess_input(x_test)
```

Una vez se tienen los conjuntos de datos preprocesados de forma correcta, se carga la red y se les pasan los conjuntos para que extraiga las características correspondientes. El conjunto de características del *train* se utilizará para entrenar la red totalmente conectada.

```
# Modelo ResNet50 (preentrenado en ImageNet y sin la última capa)
resnet50 = ResNet50(include_top=False,weights='imagenet',pooling='avg')

# Extraer las características las imágenes con el modelo anterior.
train_pred = resnet50.predict(x_train, verbose=1)
test_pred = resnet50.predict(x_test, verbose=1)
```

Por tanto, una vez se tiene las características extraídas, se monta una pequeña red totalmente conectada, como se muestra a continuación:

Para crear la red, se utilizó la última capa de la red del apartado 2, ajustando las neuronas, ya que esa red conseguía muy buenos resultados. Esa red se compila y se entrena con las características extraídas del conjunto de entrenamiento.

La arquitectura de esa segunda red sería la siguiente:

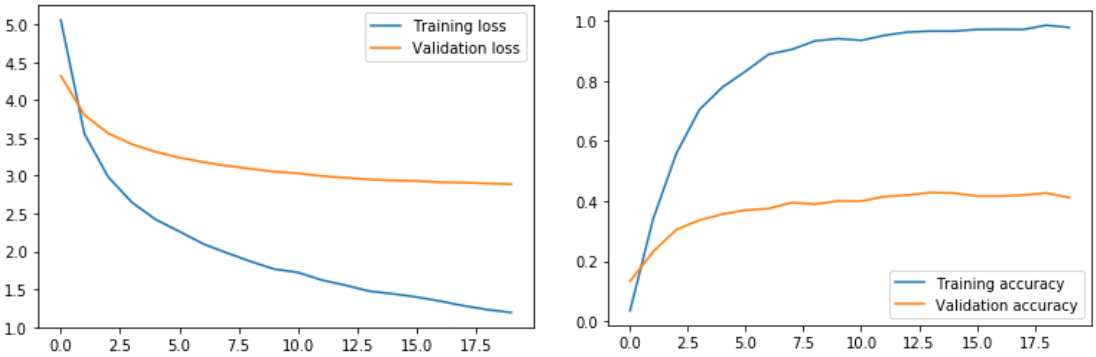
```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_33 (Dense)             (None, 2048)              4196352
_________________________________________________________________
batch_normalization_78 (Batc (None, 2048)              8192
_________________________________________________________________
activation_140 (Activation)  (None, 2048)              0
_________________________________________________________________
dense_34 (Dense)             (None, 200)               409800
_________________________________________________________________
batch_normalization_79 (Batc (None, 200)               800
_________________________________________________________________
activation_141 (Activation)  (None, 200)               0
=================================================================
Total params: 4,615,144
Trainable params: 4,610,648
Non-trainable params: 4,496
_________________________________________________________________
```

La primera capa Dense utiliza tantas neuronas como características se extrajeron con ResNet (2048) y la segunda se reduce a 200, una para cada clase que se quiere clasificar.

```
Train on 2400 samples, validate on 600 samples
Epoch 1/20
2400/2400 [==============================] - 9s 4ms/step - loss: 5.0588 - acc: 0.0354 - val_loss: 4.3226 - val_acc: 0.1333
Epoch 2/20
2400/2400 [==============================] - 3s 1ms/step - loss: 3.5525 - acc: 0.3408 - val_loss: 3.8029 - val_acc: 0.2317
Epoch 3/20
2400/2400 [==============================] - 3s 1ms/step - loss: 2.9845 - acc: 0.5575 - val_loss: 3.5610 - val_acc: 0.3050
Epoch 4/20
2400/2400 [==============================] - 3s 1ms/step - loss: 2.6486 - acc: 0.7037 - val_loss: 3.4168 - val_acc: 0.3367
Epoch 5/20
2400/2400 [==============================] - 3s 1ms/step - loss: 2.4223 - acc: 0.7788 - val_loss: 3.3163 - val_acc: 0.3567
Epoch 6/20
2400/2400 [==============================] - 3s 1ms/step - loss: 2.2643 - acc: 0.8317 - val_loss: 3.2409 - val_acc: 0.3700
Epoch 7/20
2400/2400 [==============================] - 3s 1ms/step - loss: 2.1009 - acc: 0.8879 - val_loss: 3.1808 - val_acc: 0.3750
Epoch 8/20
2400/2400 [==============================] - 3s 1ms/step - loss: 1.9811 - acc: 0.9042 - val_loss: 3.1315 - val_acc: 0.3950
Epoch 9/20
2400/2400 [==============================] - 3s 1ms/step - loss: 1.8682 - acc: 0.9325 - val_loss: 3.0925 - val_acc: 0.3900
Epoch 10/20
2400/2400 [==============================] - 3s 1ms/step - loss: 1.7670 - acc: 0.9400 - val_loss: 3.0543 - val_acc: 0.4000
Epoch 11/20
2400/2400 [==============================] - 3s 1ms/step - loss: 1.7224 - acc: 0.9342 - val_loss: 3.0328 - val_acc: 0.4000
Epoch 12/20
2400/2400 [==============================] - 3s 1ms/step - loss: 1.6216 - acc: 0.9508 - val_loss: 2.9963 - val_acc: 0.4150
Epoch 13/20
2400/2400 [==============================] - 3s 1ms/step - loss: 1.5538 - acc: 0.9617 - val_loss: 2.9738 - val_acc: 0.4200
Epoch 14/20
2400/2400 [==============================] - 3s 1ms/step - loss: 1.4760 - acc: 0.9650 - val_loss: 2.9517 - val_acc: 0.4283
Epoch 15/20
2400/2400 [==============================] - 3s 1ms/step - loss: 1.4398 - acc: 0.9650 - val_loss: 2.9384 - val_acc: 0.4267
Epoch 16/20
2400/2400 [==============================] - 3s 1ms/step - loss: 1.3969 - acc: 0.9704 - val_loss: 2.9316 - val_acc: 0.4167
Epoch 17/20
2400/2400 [==============================] - 3s 1ms/step - loss: 1.3433 - acc: 0.9708 - val_loss: 2.9138 - val_acc: 0.4167
Epoch 18/20
2400/2400 [==============================] - 3s 1ms/step - loss: 1.2816 - acc: 0.9704 - val_loss: 2.9089 - val_acc: 0.4200
Epoch 19/20
2400/2400 [==============================] - 3s 1ms/step - loss: 1.2301 - acc: 0.9846 - val_loss: 2.8987 - val_acc: 0.4267
Epoch 20/20
2400/2400 [==============================] - 3s 1ms/step - loss: 1.1921 - acc: 0.9771 - val_loss: 2.8898 - val_acc: 0.4117
3033/3033 [==============================] - 3s 938us/step
```

Al evaluar el conjunto de *test* se consigue:

- Porcentaje de *accuracy* conseguido: `36.5 %`
- Gráficas de la evolución de los valores de la función de pérdida y del *accuracy* conseguidos durante el entrenamiento del modelo:



3.2. Reentrenamiento de ResNet50

En esta segunda parte, se reentrena la red ResNet50 añadiéndole una capa totalmente conectada, como en el modelo anterior.

Para eso, se les hace el mismo preprocesamiento a los conjuntos y se crea el modelo, añadiendo las mismas capas que al modelo anterior para poder comparar los resultados y estudiar qué opción es mejor.

```
# Modelo ResNet50 con nueva capa totalmente conectada
x = resnet50.output
x = Dense(2048, activation='relu')(x)
x = BatchNormalization()(x)
last = Dense(200, activation='softmax')(x)
model = Model(inputs=resnet50.input, outputs=last)
```

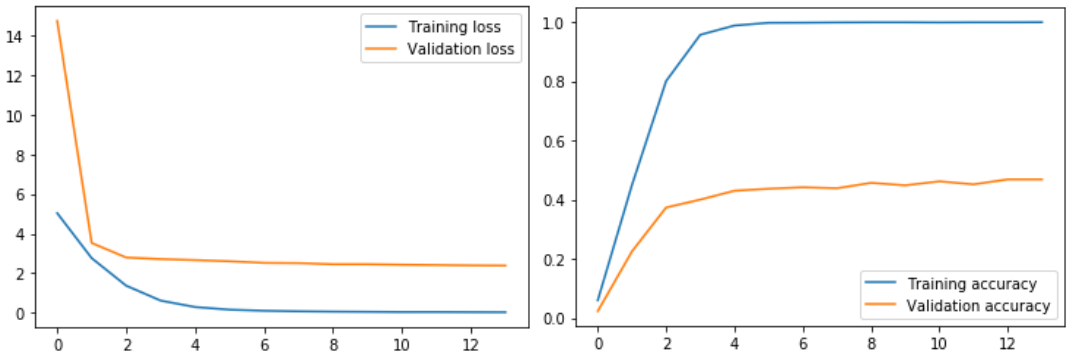
El modelo anterior se compila, como se compilaban las redes de los apartados anteriores, y se entrena con el conjunto de *train* previamente procesado. Un 20% del conjunto se utiliza como conjunto de validación.

El tiempo de ejecución es mucho mayor que el del apartado anterior, pero los resultados conseguidos son, aproximadamente un 10% mejores que los conseguidos utilizando la red como extractor de características.

```
Train on 2400 samples, validate on 600 samples
Epoch 1/14
2400/2400 [==============================] - 1266s 527ms/step - loss: 5.0306 - acc: 0.0600 - val_loss: 14.7195 - val_acc: 0.0233
Epoch 2/14
2400/2400 [==============================] - 1248s 520ms/step - loss: 2.7612 - acc: 0.4492 - val_loss: 3.5264 - val_acc: 0.2250
Epoch 3/14
2400/2400 [==============================] - 1241s 517ms/step - loss: 1.3739 - acc: 0.8004 - val_loss: 2.7929 - val_acc: 0.3733
Epoch 4/14
2400/2400 [==============================] - 1228s 512ms/step - loss: 0.6212 - acc: 0.9567 - val_loss: 2.7140 - val_acc: 0.4000
Epoch 5/14
2400/2400 [==============================] - 1246s 519ms/step - loss: 0.2962 - acc: 0.9879 - val_loss: 2.6626 - val_acc: 0.4300
Epoch 6/14
2400/2400 [==============================] - 1249s 520ms/step - loss: 0.1636 - acc: 0.9971 - val_loss: 2.6041 - val_acc: 0.4367
Epoch 7/14
2400/2400 [==============================] - 1263s 526ms/step - loss: 0.1083 - acc: 0.9975 - val_loss: 2.5229 - val_acc: 0.4417
Epoch 8/14
2400/2400 [==============================] - 1258s 524ms/step - loss: 0.0804 - acc: 0.9983 - val_loss: 2.5107 - val_acc: 0.4383
Epoch 9/14
2400/2400 [==============================] - 1261s 525ms/step - loss: 0.0646 - acc: 0.9988 - val_loss: 2.4542 - val_acc: 0.4567
Epoch 10/14
2400/2400 [==============================] - 1265s 527ms/step - loss: 0.0548 - acc: 0.9988 - val_loss: 2.4543 - val_acc: 0.4483
Epoch 11/14
2400/2400 [==============================] - 1266s 527ms/step - loss: 0.0446 - acc: 0.9983 - val_loss: 2.4312 - val_acc: 0.4617
Epoch 12/14
2400/2400 [==============================] - 1262s 526ms/step - loss: 0.0454 - acc: 0.9988 - val_loss: 2.4134 - val_acc: 0.4517
Epoch 13/14
2400/2400 [==============================] - 1271s 529ms/step - loss: 0.0394 - acc: 0.9988 - val_loss: 2.3942 - val_acc: 0.4683
Epoch 14/14
2400/2400 [==============================] - 1281s 534ms/step - loss: 0.0342 - acc: 0.9992 - val_loss: 2.3859 - val_acc: 0.4683
3033/3033 [==============================] - 484s 160ms/step
```

Al evaluar el conjunto de *test* se consigue:

- Porcentaje de *accuracy* conseguido: `40.13 %`
- Gráficas de la evolución de los valores de la función de pérdida y del *accuracy* conseguidos durante el entrenamiento del modelo:



Cómo tampoco se consigue una mejora muy grande relacionada al tiempo que se tarda en ejecutar, comparando con lo que se tarda con la primera versión, es preferible utilizar la primera opción, si se quiere un entrenamiento relativamente rápido; y la segunda versión si se prefiere mejores resultados y no importa el tiempo que tarde en entrenar.

REFERENCIAS

Al Nazi Nabil, Z. (2018, Noviembre 23). *lesion-segmentation-melanoma-tl.* Retrieved from GitHub: https://github.com/zabir-nabil/lesion-segmentation-melanoma-tl/blob/master/PH2_TL_classification.ipynb

Arunava. (2018, Diciembre 25). *Convolutional Neural Network.* Retrieved from Towards: https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05

Brownlee, J. (2017, Junio 14). *How to Get Reproducible Results with Keras.* Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/reproducible-results-neural-networks-keras/

Brownlee, J. (2018, Diciembre 10). *Use Early Stopping to Halt the Training of Neural Networks At the Right Time.* Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/

DataCamp. (n.d.). *Python For Data Science Cheat Sheet Numpy*. Retrieved from DataCamp: https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf

Debut_Kele. (2017). *Extract ResNet Feature using Keras.* Retrieved from Kaggle: https://www.kaggle.com/kelexu/extract-resnet-feature-using-keras

Dwivedi, P. (2019, Enero 4). *Understanding and Coding a ResNet in Keras.* Retrieved from Towards: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

Franky. (2018, Enero 22). *Using Keras' Pre-trained Models for Feature Extraction in Image Clustering.* Retrieved from Medium: https://medium.com/@franky07724_57962/using-keras-pre-trained-models-for-feature-extraction-in-image-clustering-a142c6cdf5b1

Keras: The Python Deep Learning library. (n.d.). Retrieved from Keras: https://keras.io

Khan, R. (2018). *How to use pre-trained models (VGG16, ResNet50...etc.) using Keras?* Retrieved from DataCamp: https://www.datacamp.com/community/news/how-to-use-pre-trained-models-vgg16-resnet50etc-using-keras-ecb6gmdr5mf

Lee, T. (2019, Marzo 29). *keras-applications.* Retrieved from GitHub: https://github.com/keras-team/keras-applications/blob/master/keras_applications/resnet50.py

Michlin, I. (2019, Octubre 5). *Keras data generators and how to use them.* Retrieved from Towards: https://towardsdatascience.com/keras-data-generators-and-how-to-use-them-b69129ed779c

Noble, R. (2017, Septiembre 11). *Comparing pre-trained deep learning models for feature extraction.* Retrieved from Zegami: https://tech.zegami.com/comparing-pre-trained-deep-learning-models-for-feature-extraction-c617da54641

Overfitting in Machine Learning: What It Is and How to Prevent It. (n.d.). Retrieved from Elite Data Science: https://elitedatascience.com/overfitting-in-machine-learning#how-to-prevent

Python keras.applications.resnet50.ResNet50() Examples. (n.d.). Retrieved from ProgramCreek: https://www.programcreek.com/python/example/100068/keras.applications.resnet50.ResNet50

Rizwan, M. (2018, Octubre 8). *Keras Application for Pre-trained Model.* Retrieved from engMRK: https://engmrk.com/kerasapplication-pre-trained-model/?utm_campaign=News&utm_medium=Community&utm_source=DataCamp.com

Rosebrock, A. (2019, Mayo 27). *Keras: Feature extraction on large datasets with Deep Learning.* Retrieved from pyImageSearch: https://www.pyimagesearch.com/2019/05/27/keras-feature-extraction-on-large-datasets-with-deep-learning/

Sharma, S. (2017, Septiembre 6). *Activation Functions in Neural Networks.* Retrieved from Towards: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Utrera Burgal, J. (2018, Julio 11). *Deep Learning básico con Keras (Parte 2): Convolutional Nets.* Retrieved from En mi local funciona: https://enmilocalfunciona.io/deep-learning-basico-con-keras-parte-2-convolutional-nets/