Laboratorio: Convolutional Neural Networks

En este laboratorio, vamos a trabajar con Convolutional Neural Networks para resolver un problen vamos a clasificar imágenes de personajes de la conocida serie de los Simpsons.

Como las CNN profundas son un tipo de modelo bastante avanzado y computacionalmente costo Google Colaboratory con soporte para GPUs. En este enlace se explica cómo activar un entorno c *estandarizarlas al mismo tamaño se usa la librería opencv. Esta Íbrería está ya instalada en el ento tendréis que instalarla.*

El dataset a utilizar consiste en imágenes de personajes de los Simpsons extraídas directamente recopilado por Alexandre Attia y es más complejo que el dataset de Fashion MNIST que hemos u clases (vamos a utilizar los 18 personajes con más imágenes), los personajes pueden aparecer e la imagen o con otros personajes en pantalla (si bien el personaje a clasificar siempre aparece en

El dataset de training puede ser descargado desde aquí:

Training data (~500MB)

Por otro lado, el dataset de test puede ser descargado de aquí:

Test data (~10MB)

Antes de empezar la práctica, se recomienda descargar las imágenes y echarlas un vistazo.

Carga de los datos

```
import cv2
import os
import numpy as np
import keras
import matplotlib.pyplot as plt
import glob
```

```
# Primero, bajamos los datos de entrenamiento
keras.utils.get_file(fname="simpsons_train.tar.gz",
                     origin="https://onedrive.live.com/download?cid=C506CF0A4F373B0F&resic

# Descomprimimos el archivo
!tar -xzf /root/.keras/datasets/simpsons_train.tar.gz -C /root/.keras/datasets

# Hacemos lo mismo con los datos de test
keras.utils.get_file(fname="simpsons_test.tar.gz",
                     origin="https://onedrive.live.com/download?cid=C506CF0A4F373B0F&resic
!tar -xzf /root/.keras/datasets/simpsons_test.tar.gz -C /root/.keras/datasets
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the %t

```
# Esta variable contiene un mapeo de número de clase a personaje.
# Utilizamos sólo los 18 personajes del dataset que tienen más imágenes.
MAP_CHARACTERS = {
    0: 'abraham_grampa_simpson', 1: 'apu_nahasapeemapetilon', 2: 'bart_simpson',
    3: 'charles_montgomery_burns', 4: 'chief_wiggum', 5: 'comic_book_guy', 6: 'edna_krabap
    7: 'homer_simpson', 8: 'kent_brockman', 9: 'krusty_the_clown', 10: 'lisa_simpson',
    11: 'marge_simpson', 12: 'milhouse_van_houten', 13: 'moe_szyslak',
    14: 'ned_flanders', 15: 'nelson_muntz', 16: 'principal_skinner', 17: 'sideshow_bob'
}

# Vamos a standarizar todas las imágenes a tamaño 64x64
IMG_SIZE = 64


def load_train_set(dirname, map_characters, verbose=True):
    """Esta función carga los datos de training en imágenes.

    Como las imágenes tienen tamaños distintas, utilizamos la librería opencv
    para hacer un resize y adaptarlas todas a tamaño IMG_SIZE x IMG_SIZE.

    Args:
        dirname: directorio completo del que leer los datos
        map_characters: variable de mapeo entre labels y personajes
        verbose: si es True, muestra información de las imágenes cargadas

    Returns:
        X, y: X es un array con todas las imágenes cargadas con tamaño
                IMG_SIZE x IMG_SIZE
              y es un array con las labels de correspondientes a cada imagen
    """
    X_train = []
    y_train = []
    for label, character in map_characters.items():
        files = os.listdir(os.path.join(dirname, character))
        images = [file for file in files if file.endswith("jpg")]
        if verbose:
          print("Leyendo {} imágenes encontradas de {}".format(len(images), character))
```

```
            for image_name in images:
                image = cv2.imread(os.path.join(dirname, character, image_name))
                X_train.append(cv2.resize(image,(IMG_SIZE, IMG_SIZE)))
                y_train.append(label)
        return np.array(X_train), np.array(y_train)


    def load_test_set(dirname, map_characters, verbose=True):
        """Esta función funciona de manera equivalente a la función load_train_set
        pero cargando los datos de test."""
        X_test = []
        y_test = []
        reverse_dict = {v: k for k, v in map_characters.items()}
        for filename in glob.glob(dirname + '/*.*'):
            char_name = "_".join(filename.split('/')[-1].split('_')[:-1])
            if char_name in reverse_dict:
                image = cv2.imread(filename)
                image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))
                X_test.append(image)
                y_test.append(reverse_dict[char_name])
        if verbose:
            print("Leídas {} imágenes de test".format(len(X_test)))
        return np.array(X_test), np.array(y_test)



    # Cargamos los datos. Si no estás trabajando en colab, cambia los paths por
    # los de los ficheros donde hayas descargado los datos.
    DATASET_TRAIN_PATH_COLAB = "/root/.keras/datasets/simpsons"
    DATASET_TEST_PATH_COLAB = "/root/.keras/datasets/simpsons_testset"

    X, y = load_train_set(DATASET_TRAIN_PATH_COLAB, MAP_CHARACTERS)
    X_t, y_t = load_test_set(DATASET_TEST_PATH_COLAB, MAP_CHARACTERS)
```

```
Leyendo 913 imágenes encontradas de abraham_grampa_simpson
Leyendo 623 imágenes encontradas de apu_nahasapeemapetilon
Leyendo 1342 imágenes encontradas de bart_simpson
Leyendo 1193 imágenes encontradas de charles_montgomery_burns
Leyendo 986 imágenes encontradas de chief_wiggum
Leyendo 469 imágenes encontradas de comic_book_guy
Leyendo 457 imágenes encontradas de edna_krabappel
Leyendo 2246 imágenes encontradas de homer_simpson
Leyendo 498 imágenes encontradas de kent_brockman
Leyendo 1206 imágenes encontradas de krusty_the_clown
Leyendo 1354 imágenes encontradas de lisa_simpson
Leyendo 1291 imágenes encontradas de marge_simpson
Leyendo 1079 imágenes encontradas de milhouse_van_houten
Leyendo 1452 imágenes encontradas de moe_szyslak
Leyendo 1454 imágenes encontradas de ned_flanders
Leyendo 358 imágenes encontradas de nelson_muntz
Leyendo 1194 imágenes encontradas de principal_skinner
Leyendo 877 imágenes encontradas de sideshow_bob
Leídas 890 imágenes de test
```

```
# Vamos a barajar aleatoriamente los datos. Esto es importante ya que si no
# lo hacemos y, por ejemplo, cogemos el 20% de los datos finales como validation
# set, estaremos utilizando solo un pequeño número de personajes, ya que
# las imágenes se leen secuencialmente personaje a personaje.
```

```
perm = np.random.permutation(len(X))
X, y = X[perm], y[perm]
print(perm)
```

```
[ 3948  9138  2210 ... 15290 16013 16823]
```

Entregable

Utilizando Convolutional Neural Networks con Keras, entrenar un clasificador que sea capaz de re
Simpsons con una accuracy en el dataset de test de **85%**. Redactar un informe analizando varias
obtenidos.

A continuación se detallan una serie de aspectos orientativos que podrían ser analizados en vues
ni mucho menos, esto son ideas orientativas de aspectos que podéis explorar):

- Análisis de los datos a utilizar.
- Análisis de resultados, obtención de métricas de *precision* y *recall* por clase y análisis de qu
resultados.
- Análisis visual de los errores de la red. ¿Qué tipo de imágenes o qué personajes dan más pr
- Comparación de modelos CNNs con un modelo de Fully Connected para este problema.
- Utilización de distintas arquitecturas CNNs, comentando aspectos como su profundidad, hi
técnicas de regularización, *batch normalization*, etc.
- [*algo más difícil*] Utilización de *data augmentation*. Esto puede conseguirse con la clase Im

Notas:

- Recuerda partir los datos en training/validation para tener una buena estimación de los valo
de test, así como comprobar que no estamos cayendo en overfitting. Una posible partición
- No es necesario mostrar en el notebook las trazas de entrenamiento de todos los modelos
guardar gráficas de esos entrenamientos para el análisis. Sin embargo, **se debe mostrar el**
obtenido y la evaluación de los datos de test con este modelo.
- Las imágenes **no están normalizadas**. Hay que normalizarlas como hemos hecho en trabaj
- El test set del problema tiene imágenes un poco más "fáciles", por lo que es posible encontr
mejores que en el training set.

Librerias utilies

```
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, BatchNormalization
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Activation
from keras.utils import plot_model
from keras.preprocessing.image import ImageDataGenerator
```

Preprocesado

```
def namestr(obj, namespace):
    mstr = [name for name in namespace if namespace[name] is obj][0]
    return mstr

def printshapes(data):
    mstr = namestr(data, globals())
    print(str(mstr) + ' shape = \t',data.shape)


# Preprocesado

# variables
xtest = X_t
ytest = y_t

#Escalado
x = X / 255.0
xtest = X_t / 255.0

# number of clases
N = len(np.unique(y))
print('NUmero de clases = ',N)

# dividimos train en train/val
perc = 0.2
xtrain, xval, ytrain, yval = train_test_split(x, y, test_size=perc)

# convertimos las salida a one hot
ytrain = keras.utils.to_categorical(ytrain,N)
yval = keras.utils.to_categorical(yval,N)
ytest = keras.utils.to_categorical(ytest,N)

### ________________________________________________
###  Reshape (solo para Fully Connected)

# actualizamos dimensiones
N1,X1,Y1,C1 =  xtrain.shape
N2,X2,Y2,C2 =  xval.shape
N3,X3,Y3,C3 =  xtest.shape

#hacemos reshape de la entrada
xtrain_grey = np.dot(xtrain[...,:3], [0.2989, 0.5870, 0.1140])
xval_grey = np.dot(xval[...,:3], [0.2989, 0.5870, 0.1140])
xtest_grey = np.dot(xtest[...,:3], [0.2989, 0.5870, 0.1140])

xtrainfc  = xtrain_grey.reshape(N1,X1*Y1)
xvalfc    = xval_grey.reshape(N2,X2*Y2)
xtestfc   = xtest_grey.reshape(N3,X3*Y3)

ytrainfc  = ytrain
yvalfc    = yval
```

```
ytestfc   = ytest

### ____________________________________________


# Shapes2
vsh = [xtrain,xval,ytrain,yval,xtest,ytest]

for i in vsh:
  printshapes(i)
```

```
NUmero de clases =  18
xtrain shape =   (15193, 64, 64, 3)
xval shape =     (3799, 64, 64, 3)
ytrain shape =   (15193, 18)
yval shape =     (3799, 18)
xtest shape =    (890, 64, 64, 3)
ytest shape =    (890, 18)
```

```
xtrainfc.shape
```

```
(15193, 4096)
```

Sintesis

```
# definimos los parametros del modelo, datos
inputdim_fc = xtrainfc.shape[1]
print('Input Dim FC = ', inputdim_fc)

inputdim_cnn = xtrain.shape[1:]
print('Input Dim CNN = ', inputdim_cnn)
```

```
Input Dim FC =  4096
Input Dim CNN =  (64, 64, 3)
```

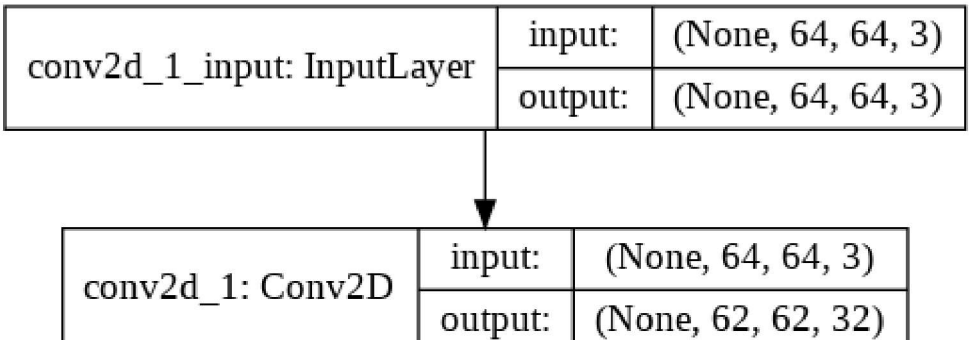
Modelo 1 -- CNN (I)

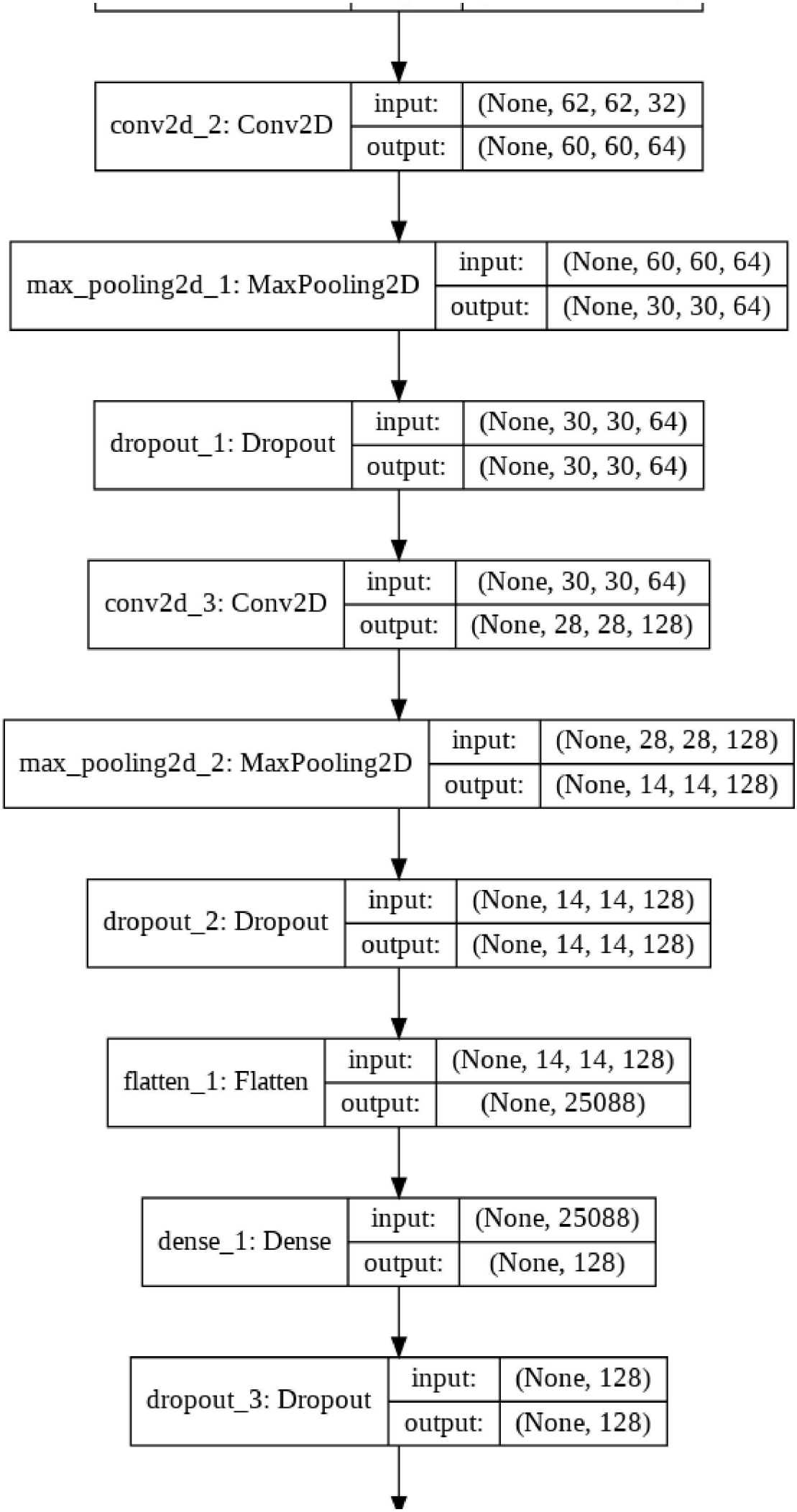
3 CNN + 2 Dense (Kernel 3)

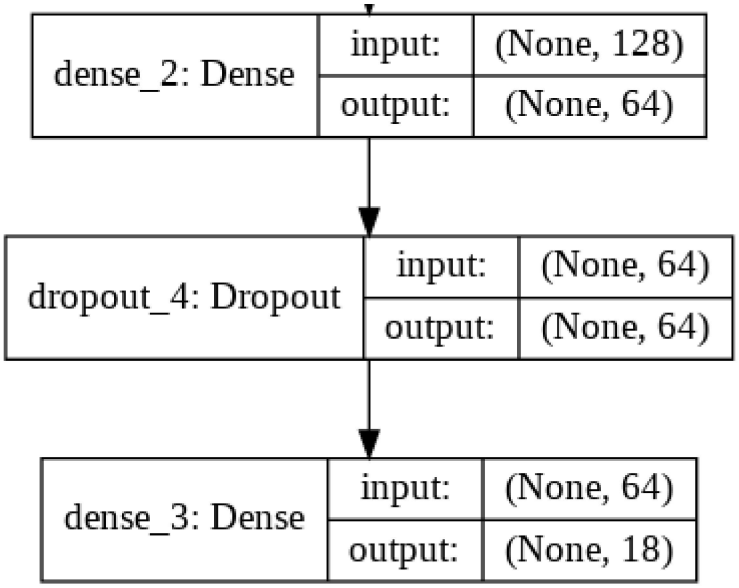
```
# Modelo 1

#layers
f1 = 32
f2 = 64
f3 = 128

l1 = 128
l2 = 64

#batch
```

```
mepochs = 30
mbatch = 32

# kernel
k = 3

#activation
act = 'relu'

# pooling
p=2
ps = (p,p)

# dropout
drop = 0.2

## CNN

model1 = Sequential()
model1.add(Conv2D(f1, kernel_size=(k,k),activation=act,input_shape=inputdim_cnn))

model1.add(Conv2D(f2, kernel_size=(k,k),activation=act))
model1.add(MaxPooling2D(pool_size=ps))
model1.add(Dropout(drop))

model1.add(Conv2D(f3, kernel_size=(k,k),activation=act))
model1.add(MaxPooling2D(pool_size=ps))
model1.add(Dropout(drop))

# DENSE
model1.add(Flatten()) #coge el volumen -> vector

model1.add(Dense(l1,activation = act))
model1.add(Dropout(drop))

model1.add(Dense(l2,activation = act))
model1.add(Dropout(drop))

model1.add(Dense(N,activation = 'softmax'))

 # compilamos el modelo
model1.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])


# Summary
model1.summary()

i = str(1) #numero de model1o
plot_model(model1, show_shapes = 'True', to_file='model1'+i+'.png')
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:79

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 62, 62, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 60, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 14, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               3211392
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 18)                1170
=================================================================
Total params: 3,314,066
Trainable params: 3,314,066
Non-trainable params: 0
_________________________________________________________________
```




conv2d_2: Conv2D
input: (None, 62, 62, 32)
output: (None, 60, 60, 64)
max_pooling2d_1: MaxPooling2D
input: (None, 60, 60, 64)
output: (None, 30, 30, 64)
dropout_1: Dropout
input: (None, 30, 30, 64)
output: (None, 30, 30, 64)
conv2d_3: Conv2D
input: (None, 30, 30, 64)
output: (None, 28, 28, 128)
max_pooling2d_2: MaxPooling2D
input: (None, 28, 28, 128)
output: (None, 14, 14, 128)
dropout_2: Dropout
input: (None, 14, 14, 128)
output: (None, 14, 14, 128)
flatten_1: Flatten
input: (None, 14, 14, 128)
output: (None, 25088)
dense_1: Dense
input: (None, 25088)
output: (None, 128)
dropout_3: Dropout
input: (None, 128)
output: (None, 128)

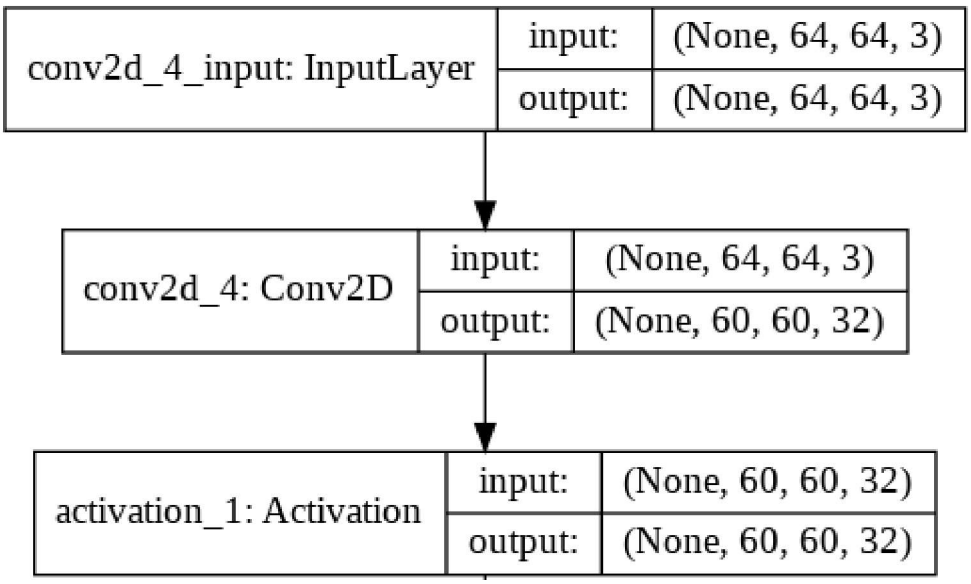


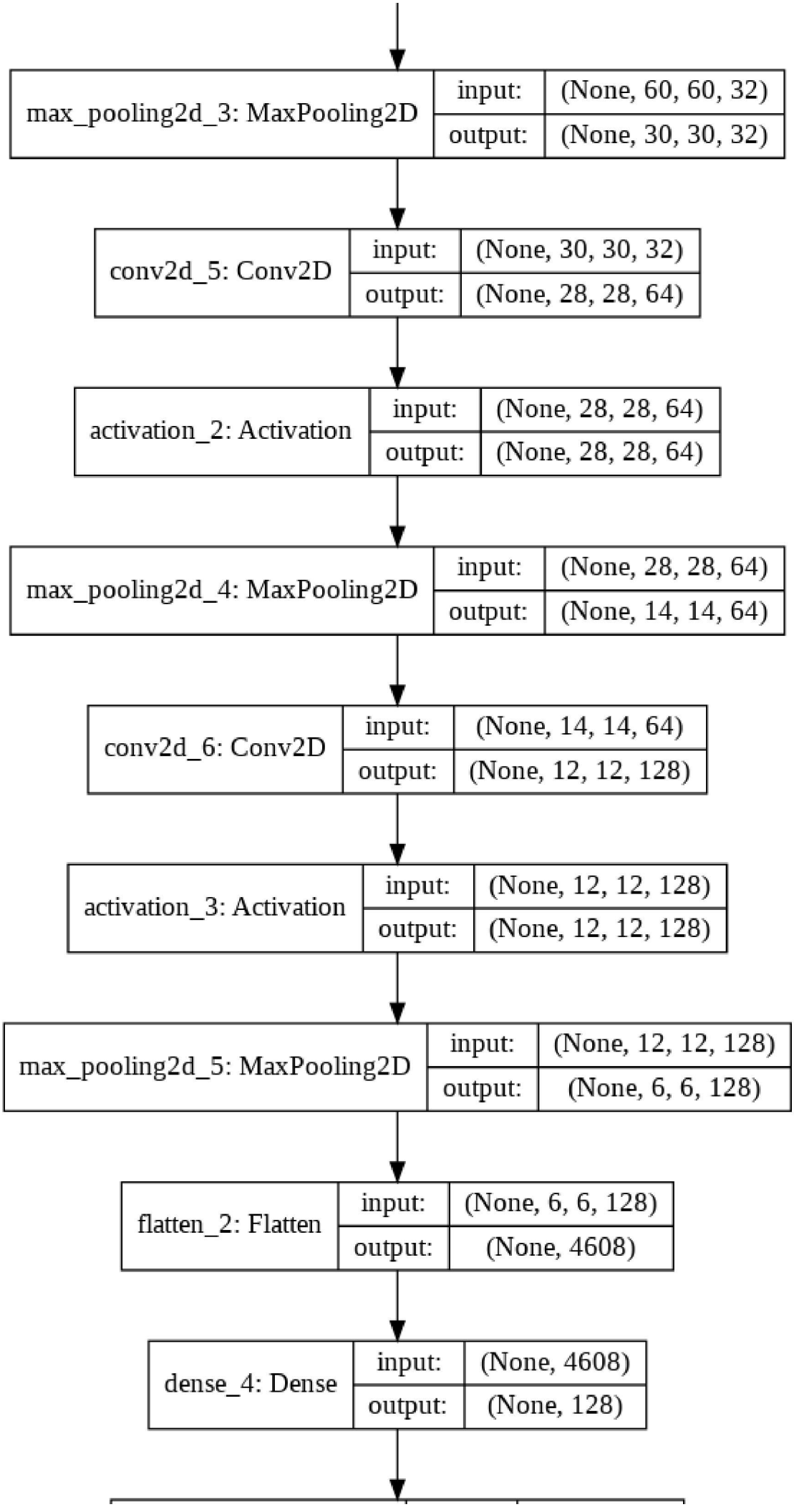
Modelo 2 -- CNN (II)

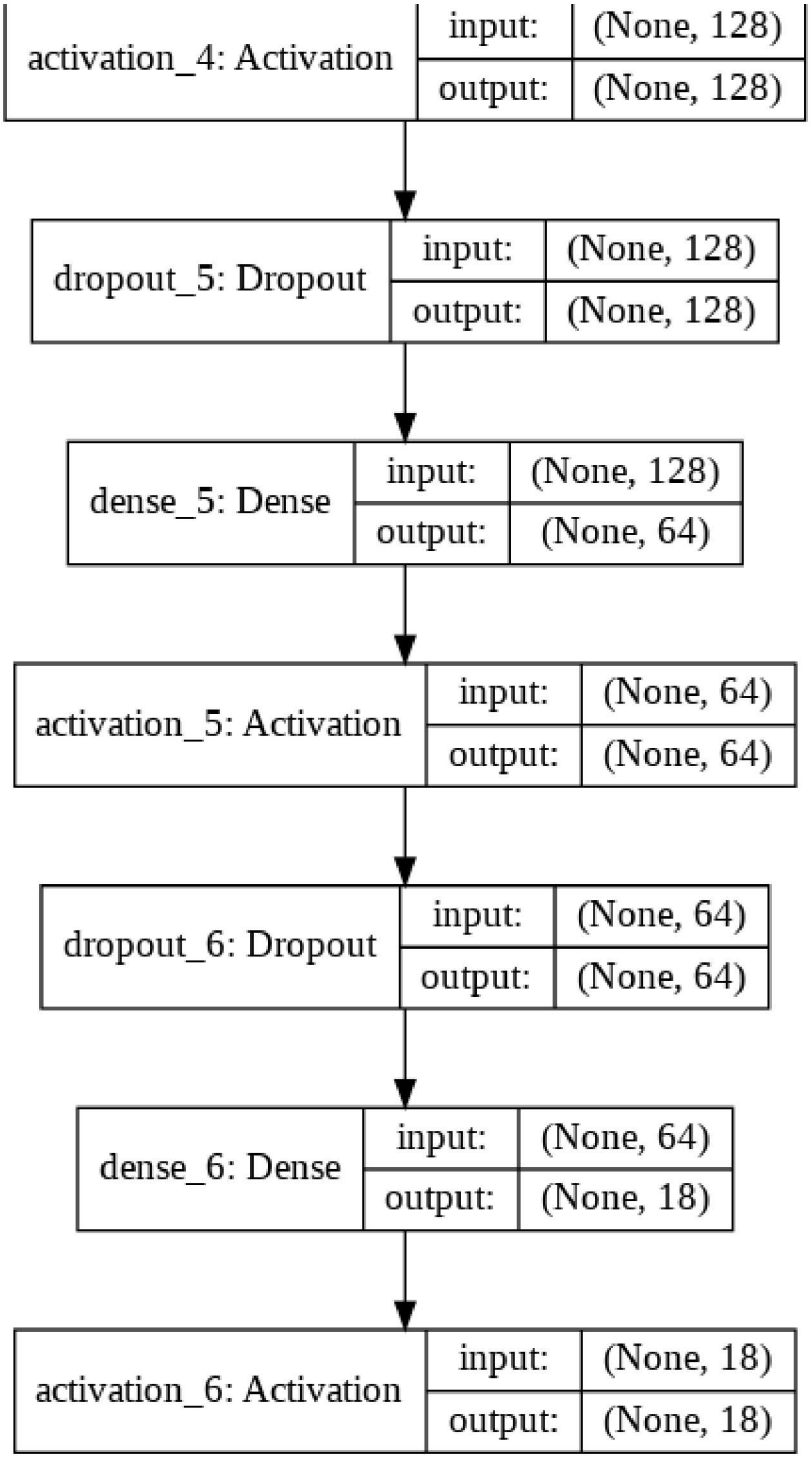
3 CNN + 2 Dense (Kernel 5/3)

```
# model2o 2
model2 = Sequential()
model2.add(Conv2D(32, (5, 5), input_shape=inputdim_cnn))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Conv2D(64, (3, 3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Conv2D(128, (3, 3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Flatten())
model2.add(Dense(128))
model2.add(Activation('relu'))
model2.add(Dropout(0.5))
model2.add(Dense(64))
model2.add(Activation('relu'))
model2.add(Dropout(0.5))
model2.add(Dense(N))
model2.add(Activation('softmax'))

# compilamos el modelo
model2.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])

model2.summary()

i = str(2) #numero de model2o
plot_model(model2, show_shapes = 'True', to_file='model2'+i+'.png')
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 60, 60, 32)        2432
_________________________________________________________________
activation_1 (Activation)    (None, 60, 60, 32)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 30, 30, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 28, 28, 64)        0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 12, 12, 128)       73856
_________________________________________________________________
activation_3 (Activation)    (None, 12, 12, 128)       0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               589952
_________________________________________________________________
activation_4 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 64)                8256
_________________________________________________________________
activation_5 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 18)                1170
_________________________________________________________________
activation_6 (Activation)    (None, 18)                0
=================================================================
Total params: 694,162
Trainable params: 694,162
Non-trainable params: 0
_________________________________________________________________
```




max_pooling2d_3: MaxPooling2D
input: (None, 60, 60, 32)
output: (None, 30, 30, 32)
conv2d_5: Conv2D
input: (None, 30, 30, 32)
output: (None, 28, 28, 64)
activation_2: Activation
input: (None, 28, 28, 64)
output: (None, 28, 28, 64)
max_pooling2d_4: MaxPooling2D
input: (None, 28, 28, 64)
output: (None, 14, 14, 64)
conv2d_6: Conv2D
input: (None, 14, 14, 64)
output: (None, 12, 12, 128)
activation_3: Activation
input: (None, 12, 12, 128)
output: (None, 12, 12, 128)
max_pooling2d_5: MaxPooling2D
input: (None, 12, 12, 128)
output: (None, 6, 6, 128)
flatten_2: Flatten
input: (None, 6, 6, 128)
output: (None, 4608)
dense_4: Dense
input: (None, 4608)
output: (None, 128)



Modelo 3 -- VGG 19

```
model3 = keras.applications.vgg19.VGG19(include_top=True,
                                        weights=None,
                                        input_tensor=None,
                                        input_shape=inputdim_cnn,
                                        pooling=None,
                                        classes=N)
 # compilamos el modelo
model3.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])

model3.summary()
```

```
Model: "vgg19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 64, 64, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              8392704
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 18)                73746
=================================================================
Total params: 45,272,146
Trainable params: 45,272,146
Non-trainable params: 0
```

Modelo 4 - Data Augmentation

(construimos solo los datos y al compilar cambiamos el name del modelo)

```
# ampliamos los datos
train_datagen = ImageDataGenerator(horizontal_flip=True)
test_datagen = ImageDataGenerator()

train_generator = train_datagen.flow(xtrain, ytrain)
validation_generator = test_datagen.flow(xval, yval)

model4 = Sequential()
model4.add(Conv2D(f1, kernel_size=(k,k),activation=act,input_shape=inputdim_cnn))

model4.add(Conv2D(f2, kernel_size=(k,k),activation=act))
model4.add(MaxPooling2D(pool_size=ps))
model4.add(Dropout(drop))

model4.add(Conv2D(f3, kernel_size=(k,k),activation=act))
model4.add(MaxPooling2D(pool_size=ps))
model4.add(Dropout(drop))

# DENSE
model4.add(Flatten()) #coge el volumen -> vector

model4.add(Dense(l1,activation = act))
model4.add(Dropout(drop))

model4.add(Dense(l2,activation = act))
model4.add(Dropout(drop))

model4.add(Dense(N,activation = 'softmax'))

 # compilamos el model4o
model4.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])

# Summary
model4.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 62, 62, 32)        896
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 60, 60, 64)        18496
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_7 (Dropout)          (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
dropout_8 (Dropout)          (None, 14, 14, 128)       0
_________________________________________________________________
flatten_3 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               3211392
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_8 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_10 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_9 (Dense)              (None, 18)                1170
=================================================================
Total params: 3,314,066
Trainable params: 3,314,066
Non-trainable params: 0
_________________________________________________________________
```

Modelo 5 - VGG 19 (II) + DA

modificando FC Layers Data augmentation

```
model_base= keras.applications.vgg19.VGG19(include_top=False,
                                           weights='imagenet',
                                           input_tensor=None,
                                           input_shape=inputdim_cnn,
                                           pooling=None,
                                           classes=N)

model_base.summary()
print('\n')


x = model_base.output
x = Flatten()(x)
x = BatchNormalization()(x)

x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
```

```
x = BatchNormalization()(x)

x = Dense(64, activation='relu')(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)

x = Dense(N, activation='softmax')(x)

model5 = Model(inputs=model_base.input, outputs=x)

 # compilamos el modelo
model5.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])

model5.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

Model: "vgg19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 64, 64, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0
=================================================================
Total params: 20,024,384
```

```
Trainable params: 20,024,384
Non-trainable params: 0
_________________________________________________________________


Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 64, 64, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 2048)              0
_________________________________________________________________
batch_normalization_1 (Batch (None, 2048)              8192
_________________________________________________________________
dense_10 (Dense)             (None, 128)               262272
_________________________________________________________________
dropout_11 (Dropout)         (None, 128)               0
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512
```

```
batch_normalization_2 (Batch (None, 128)               512
_________________________________________________________________
dense_11 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_12 (Dropout)         (None, 64)                0
_________________________________________________________________
batch_normalization_3 (Batch (None, 64)                256
_________________________________________________________________
dense_12 (Dense)             (None, 18)                1170
=================================================================
Total params: 20,305,042
Trainable params: 20,300,562
Non-trainable params: 4,480
_________________________________________________________________
```

Modelo 6 - Fully Conected

```
### ------------------- Eleccion de parametros --------------------
#* Número de capas y neuronas por capa
nlayer1 = 512
nlayer2 = 256
nlayer3 = 25

mepochs = 30

#* Optimizadores y sus parámetros
mlr = 0.02
opt = 'Adagrad'
#* Batch size
mbatch = 100
print('vol =',mepochs*mbatch)

#* Unidades de activación
act = 'relu'
#act = keras.layers.LeakyReLU(alpha=0.1)


#* Uso de capas dropout, regularización L2, regularización L1...
# --> Incluido en la creacion del modelo
mdrop = 0.1

#* Early stopping
from keras.callbacks import ModelCheckpoint, EarlyStopping
es = EarlyStopping(monitor='val_acc', mode='auto', verbose=1) #callback

#* Batch normalization
# --> Incluido en la creacion del modelo
# ----------------------------------------------------------------


# ------------------  creamos el modelo 8 -----------------------
model6 = Sequential()

model6.add(Dense(nlayer1,input_dim=inputdim_fc,activation = act))
model6.add(keras.layers.BatchNormalization())
```

```
model6.add(Dropout(mdrop))

model6.add(Dense(nlayer2,activation = act))
model6.add(keras.layers.BatchNormalization())
model6.add(Dropout(mdrop))

model6.add(Dense(nlayer3, activation = act))
model6.add(keras.layers.BatchNormalization())
model6.add(Dropout(mdrop))

model6.add(Dense(N, activation='softmax'))

# compilamos el modelo
model6.compile(loss='categorical_crossentropy', optimizer= opt, metrics=['accuracy'])

# resumen y grafico del modelo
model6.summary()
```

```
vol = 3000
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_13 (Dense)             (None, 512)               2097664
_________________________________________________________________
batch_normalization_4 (Batch (None, 512)               2048
_________________________________________________________________
dropout_13 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_14 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_5 (Batch (None, 256)               1024
_________________________________________________________________
dropout_14 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_15 (Dense)             (None, 25)                6425
_________________________________________________________________
batch_normalization_6 (Batch (None, 25)                100
_________________________________________________________________
dropout_15 (Dropout)         (None, 25)                0
_________________________________________________________________
dense_16 (Dense)             (None, 18)                468
=================================================================
Total params: 2,239,057
Trainable params: 2,237,471
Non-trainable params: 1,586
_________________________________________________________________
```

Train

```
#-------------------------------------------------------------------------
def alb_train(model,xtrain,ytrain,mbatch,mepochs,xval,yval):
  # Early stopping
  from keras.callbacks import ModelCheckpoint, EarlyStopping
  es = EarlyStopping(monitor='val_acc', mode='auto',  patience=4, verbose=1) #callback
```

```
  # entrenamos
  history = model.fit(xtrain, ytrain,
                      batch_size=mbatch,
                      epochs=mepochs,
                      verbose=2,
                      validation_data=(xval,yval),
                      callbacks=[es])
  return history

#------------------------------------------------------------------------
def alb_train_Data_Aug(model,train_generator,validation_generator,mbatch,mepochs):
  # Early stopping
  from keras.callbacks import ModelCheckpoint, EarlyStopping
  es = EarlyStopping(monitor='val_acc', mode='auto', patience=4, verbose=1) #callback

   # steps_per_epoch=len(xtrain) // mbatch,
  history = model.fit_generator(train_generator,
                        verbose = 2,
                        epochs=mepochs,
                        validation_data=validation_generator,
                        callbacks=[es])

  return history


print('\nModel 1 -------------')
history1 = alb_train(model1,xtrain,ytrain,mbatch,mepochs,xval,yval)
print('\nModel 2 -------------')
history2 = alb_train(model2,xtrain,ytrain,mbatch,mepochs,xval,yval)
print('\nModel 3 -------------')
history3 = alb_train(model3,xtrain,ytrain,mbatch,mepochs,xval,yval)
print('\nModel 4 -------------')
history4 = alb_train_Data_Aug(model4,train_generator,validation_generator,mbatch,mepochs)
print('\nModel 5 -------------')
history5 = alb_train(model5,xtrain,ytrain,mbatch,mepochs,xval,yval)
print('\nModel 6 -------------')
history6 = alb_train(model6,xtrainfc,ytrainfc,mbatch,mepochs,xvalfc,yvalfc)
```

```
Model 1 -------------
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/pythor
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

Train on 15193 samples, validate on 3799 samples
Epoch 1/30
 - 7s - loss: 2.5333 - acc: 0.2043 - val_loss: 2.0482 - val_acc: 0.3698
Epoch 2/30
 - 4s - loss: 1.9052 - acc: 0.4212 - val_loss: 1.5303 - val_acc: 0.5609
Epoch 3/30
 - 4s - loss: 1.4426 - acc: 0.5688 - val_loss: 1.1779 - val_acc: 0.6410
Epoch 4/30
 - 4s - loss: 1.1505 - acc: 0.6520 - val_loss: 1.0179 - val_acc: 0.6936
Epoch 5/30
 - 4s - loss: 0.9228 - acc: 0.7175 - val_loss: 0.8706 - val_acc: 0.7376
Epoch 6/30
 - 4s - loss: 0.7626 - acc: 0.7622 - val_loss: 0.7735 - val_acc: 0.7665
Epoch 7/30
 - 4s - loss: 0.6423 - acc: 0.8000 - val_loss: 0.7746 - val_acc: 0.7736
Epoch 8/30
 - 4s - loss: 0.5434 - acc: 0.8270 - val_loss: 0.7388 - val_acc: 0.7886
Epoch 9/30
 - 4s - loss: 0.4650 - acc: 0.8538 - val_loss: 0.7594 - val_acc: 0.7936
Epoch 10/30
 - 4s - loss: 0.4032 - acc: 0.8721 - val_loss: 0.7326 - val_acc: 0.7984
Epoch 11/30
 - 4s - loss: 0.3625 - acc: 0.8848 - val_loss: 0.7501 - val_acc: 0.8015
Epoch 12/30
 - 4s - loss: 0.3210 - acc: 0.8988 - val_loss: 0.7643 - val_acc: 0.7981
Epoch 13/30
 - 4s - loss: 0.3084 - acc: 0.8983 - val_loss: 0.7565 - val_acc: 0.8052
Epoch 14/30
 - 4s - loss: 0.2668 - acc: 0.9129 - val_loss: 0.7425 - val_acc: 0.8173
Epoch 15/30
 - 4s - loss: 0.2574 - acc: 0.9171 - val_loss: 0.7238 - val_acc: 0.8168
Epoch 16/30
 - 4s - loss: 0.2268 - acc: 0.9265 - val_loss: 0.8139 - val_acc: 0.8084
Epoch 17/30
 - 4s - loss: 0.2262 - acc: 0.9268 - val_loss: 0.7295 - val_acc: 0.8221
Epoch 18/30
 - 4s - loss: 0.2075 - acc: 0.9344 - val_loss: 0.7537 - val_acc: 0.8155
Epoch 19/30
 - 4s - loss: 0.1874 - acc: 0.9392 - val_loss: 0.7858 - val_acc: 0.8165
Epoch 20/30
 - 4s - loss: 0.1828 - acc: 0.9448 - val_loss: 0.7679 - val_acc: 0.8205
Epoch 21/30
 - 4s - loss: 0.1717 - acc: 0.9456 - val_loss: 0.7856 - val_acc: 0.8218
Epoch 00021: early stopping

Model 2 -------------
Train on 15193 samples, validate on 3799 samples
Epoch 1/30
 - 3s - loss: 2.7806 - acc: 0.1139 - val_loss: 2.5264 - val_acc: 0.2406
Epoch 2/30
 - 2s - loss: 2.4195 - acc: 0.2413 - val_loss: 2.0621 - val_acc: 0.3604
Epoch 3/30
```

```
 - 2s - loss: 2.0734 - acc: 0.3381 - val_loss: 1.7059 - val_acc: 0.4993
Epoch 4/30
 - 2s - loss: 1.8250 - acc: 0.4232 - val_loss: 1.5279 - val_acc: 0.5607
Epoch 5/30
 - 2s - loss: 1.5828 - acc: 0.4961 - val_loss: 1.2860 - val_acc: 0.6199
Epoch 6/30
 - 2s - loss: 1.3680 - acc: 0.5660 - val_loss: 1.0759 - val_acc: 0.6749
Epoch 7/30
 - 2s - loss: 1.2177 - acc: 0.6177 - val_loss: 0.9772 - val_acc: 0.7060
Epoch 8/30
 - 2s - loss: 1.0834 - acc: 0.6560 - val_loss: 0.8819 - val_acc: 0.7410
Epoch 9/30
 - 2s - loss: 0.9632 - acc: 0.6947 - val_loss: 0.8211 - val_acc: 0.7528
Epoch 10/30
 - 2s - loss: 0.8725 - acc: 0.7164 - val_loss: 0.7616 - val_acc: 0.7718
Epoch 11/30
 - 2s - loss: 0.7996 - acc: 0.7421 - val_loss: 0.7431 - val_acc: 0.7744
Epoch 12/30
 - 2s - loss: 0.7105 - acc: 0.7716 - val_loss: 0.6956 - val_acc: 0.7992
Epoch 13/30
 - 2s - loss: 0.6726 - acc: 0.7794 - val_loss: 0.6859 - val_acc: 0.8055
Epoch 14/30
 - 2s - loss: 0.6306 - acc: 0.7950 - val_loss: 0.6472 - val_acc: 0.8210
Epoch 15/30
 - 2s - loss: 0.5673 - acc: 0.8122 - val_loss: 0.6690 - val_acc: 0.8097
Epoch 16/30
 - 2s - loss: 0.5286 - acc: 0.8244 - val_loss: 0.6167 - val_acc: 0.8260
Epoch 17/30
 - 2s - loss: 0.5120 - acc: 0.8298 - val_loss: 0.6165 - val_acc: 0.8342
Epoch 18/30
 - 2s - loss: 0.4836 - acc: 0.8391 - val_loss: 0.6210 - val_acc: 0.8342
Epoch 19/30
 - 2s - loss: 0.4481 - acc: 0.8522 - val_loss: 0.6085 - val_acc: 0.8371
Epoch 20/30
 - 2s - loss: 0.4323 - acc: 0.8564 - val_loss: 0.6293 - val_acc: 0.8352
Epoch 21/30
 - 2s - loss: 0.4039 - acc: 0.8690 - val_loss: 0.6392 - val_acc: 0.8421
Epoch 22/30
 - 2s - loss: 0.3854 - acc: 0.8720 - val_loss: 0.6063 - val_acc: 0.8505
Epoch 23/30
 - 2s - loss: 0.3646 - acc: 0.8787 - val_loss: 0.6038 - val_acc: 0.8471
Epoch 24/30
 - 2s - loss: 0.3576 - acc: 0.8845 - val_loss: 0.6244 - val_acc: 0.8471
Epoch 25/30
 - 2s - loss: 0.3359 - acc: 0.8925 - val_loss: 0.6229 - val_acc: 0.8513
Epoch 26/30
 - 2s - loss: 0.3243 - acc: 0.8967 - val_loss: 0.6444 - val_acc: 0.8523
Epoch 27/30
 - 2s - loss: 0.3214 - acc: 0.8961 - val_loss: 0.6445 - val_acc: 0.8468
Epoch 28/30
 - 2s - loss: 0.3079 - acc: 0.9005 - val_loss: 0.6585 - val_acc: 0.8465
Epoch 29/30
 - 2s - loss: 0.2926 - acc: 0.9048 - val_loss: 0.6040 - val_acc: 0.8644
Epoch 30/30
 - 2s - loss: 0.2818 - acc: 0.9107 - val_loss: 0.6265 - val_acc: 0.8526

Model 3 -------------
Train on 15193 samples, validate on 3799 samples
Epoch 1/30
 - 20s - loss: 2.8662 - acc: 0.1075 - val_loss: 2.7864 - val_acc: 0.1253
Epoch 2/30
 - 16s - loss: 2.8010 - acc: 0.1165 - val_loss: 2.7895 - val_acc: 0.1253
```

```
Epoch 3/30
 - 16s - loss: 2.8006 - acc: 0.1165 - val_loss: 2.7941 - val_acc: 0.1253
Epoch 4/30
 - 16s - loss: 2.8003 - acc: 0.1165 - val_loss: 2.7869 - val_acc: 0.1253
Epoch 5/30
 - 16s - loss: 2.8006 - acc: 0.1165 - val_loss: 2.7882 - val_acc: 0.1253
Epoch 00005: early stopping

Model 4 -------------
Epoch 1/30
 - 7s - loss: 2.3322 - acc: 0.2787 - val_loss: 1.7604 - val_acc: 0.4859
Epoch 2/30
 - 6s - loss: 1.6626 - acc: 0.4944 - val_loss: 1.2899 - val_acc: 0.6038
Epoch 3/30
 - 6s - loss: 1.3284 - acc: 0.5956 - val_loss: 1.0601 - val_acc: 0.6857
Epoch 4/30
 - 6s - loss: 1.0880 - acc: 0.6687 - val_loss: 0.9783 - val_acc: 0.7147
Epoch 5/30
 - 6s - loss: 0.9274 - acc: 0.7188 - val_loss: 0.8135 - val_acc: 0.7591
Epoch 6/30
 - 6s - loss: 0.7762 - acc: 0.7643 - val_loss: 0.7203 - val_acc: 0.7889
Epoch 7/30
 - 6s - loss: 0.6828 - acc: 0.7926 - val_loss: 0.6886 - val_acc: 0.7968
Epoch 8/30
 - 6s - loss: 0.6182 - acc: 0.8146 - val_loss: 0.6226 - val_acc: 0.8181
Epoch 9/30
 - 6s - loss: 0.5475 - acc: 0.8319 - val_loss: 0.6020 - val_acc: 0.8278
Epoch 10/30
 - 6s - loss: 0.4981 - acc: 0.8474 - val_loss: 0.5884 - val_acc: 0.8268
Epoch 11/30
 - 6s - loss: 0.4409 - acc: 0.8639 - val_loss: 0.5798 - val_acc: 0.8300
Epoch 12/30
 - 6s - loss: 0.4152 - acc: 0.8755 - val_loss: 0.5665 - val_acc: 0.8376
Epoch 13/30
 - 6s - loss: 0.3759 - acc: 0.8842 - val_loss: 0.5682 - val_acc: 0.8444
Epoch 14/30
 - 6s - loss: 0.3469 - acc: 0.8913 - val_loss: 0.5688 - val_acc: 0.8373
Epoch 15/30
 - 6s - loss: 0.3369 - acc: 0.8983 - val_loss: 0.5289 - val_acc: 0.8523
Epoch 16/30
 - 6s - loss: 0.3044 - acc: 0.9059 - val_loss: 0.5427 - val_acc: 0.8547
Epoch 17/30
 - 6s - loss: 0.2826 - acc: 0.9139 - val_loss: 0.5581 - val_acc: 0.8579
Epoch 18/30
 - 6s - loss: 0.2675 - acc: 0.9163 - val_loss: 0.5282 - val_acc: 0.8639
Epoch 19/30
 - 6s - loss: 0.2529 - acc: 0.9237 - val_loss: 0.5662 - val_acc: 0.8531
Epoch 20/30
 - 6s - loss: 0.2456 - acc: 0.9288 - val_loss: 0.5114 - val_acc: 0.8613
Epoch 21/30
 - 6s - loss: 0.2316 - acc: 0.9298 - val_loss: 0.5347 - val_acc: 0.8642
Epoch 22/30
 - 6s - loss: 0.2091 - acc: 0.9344 - val_loss: 0.5207 - val_acc: 0.8705
Epoch 23/30
 - 6s - loss: 0.2260 - acc: 0.9306 - val_loss: 0.5349 - val_acc: 0.8586
Epoch 24/30
 - 6s - loss: 0.2034 - acc: 0.9383 - val_loss: 0.5576 - val_acc: 0.8634
Epoch 25/30
 - 6s - loss: 0.1859 - acc: 0.9435 - val_loss: 0.5106 - val_acc: 0.8721
Epoch 26/30
 - 6s - loss: 0.1925 - acc: 0.9413 - val_loss: 0.5697 - val_acc: 0.8739
```

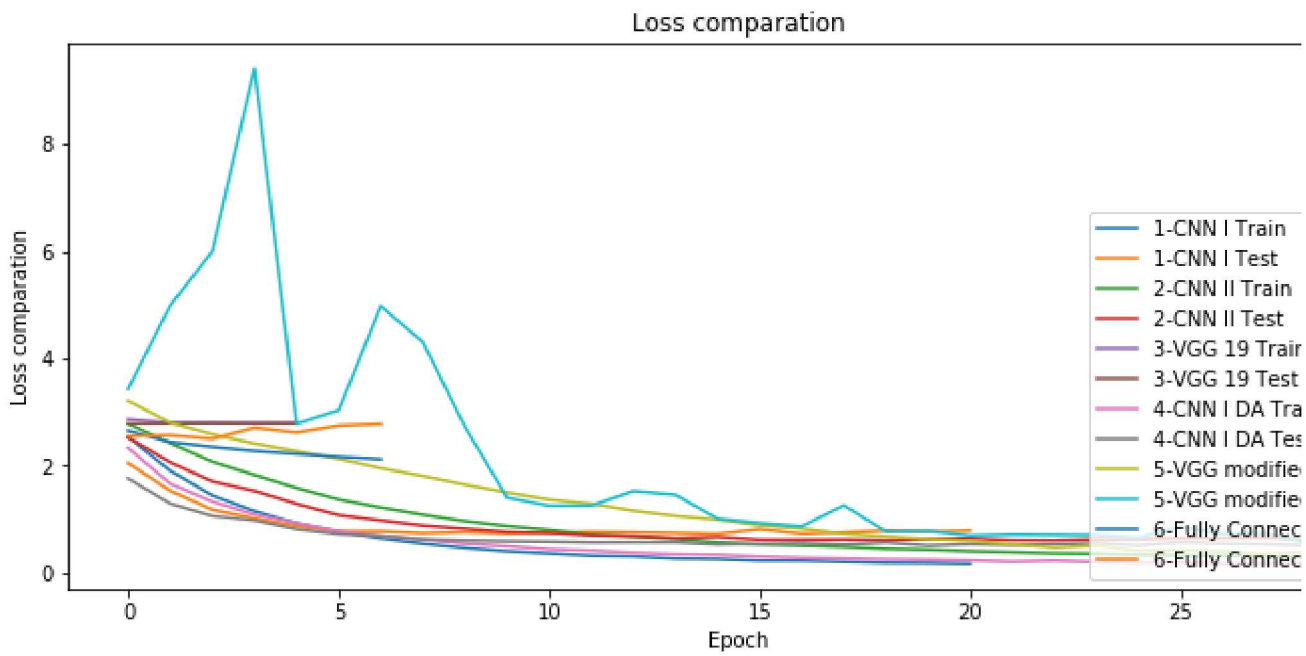
```
Epoch 27/30
 - 6s - loss: 0.1740 - acc: 0.9486 - val_loss: 0.5451 - val_acc: 0.8731
Epoch 28/30
 - 6s - loss: 0.1755 - acc: 0.9465 - val_loss: 0.5341 - val_acc: 0.8755
Epoch 29/30
 - 6s - loss: 0.1813 - acc: 0.9454 - val_loss: 0.5118 - val_acc: 0.8810
Epoch 30/30
 - 6s - loss: 0.1625 - acc: 0.9514 - val_loss: 0.5774 - val_acc: 0.8726

Model 5 -------------
Train on 15193 samples, validate on 3799 samples
Epoch 1/30
 - 18s - loss: 3.2085 - acc: 0.0812 - val_loss: 3.4309 - val_acc: 0.0824
Epoch 2/30
 - 16s - loss: 2.8014 - acc: 0.1444 - val_loss: 4.9875 - val_acc: 0.0463
Epoch 3/30
 - 16s - loss: 2.5870 - acc: 0.1937 - val_loss: 5.9981 - val_acc: 0.0461
Epoch 4/30
 - 16s - loss: 2.4106 - acc: 0.2249 - val_loss: 9.4164 - val_acc: 0.1113
Epoch 5/30
 - 16s - loss: 2.2701 - acc: 0.2555 - val_loss: 2.7824 - val_acc: 0.1882
Epoch 6/30
 - 16s - loss: 2.1251 - acc: 0.2949 - val_loss: 3.0327 - val_acc: 0.1082
Epoch 7/30
 - 16s - loss: 1.9535 - acc: 0.3573 - val_loss: 4.9771 - val_acc: 0.0792
Epoch 8/30
 - 16s - loss: 1.8046 - acc: 0.4068 - val_loss: 4.3052 - val_acc: 0.1637
Epoch 9/30
 - 16s - loss: 1.6454 - acc: 0.4633 - val_loss: 2.7228 - val_acc: 0.2137
Epoch 10/30
 - 16s - loss: 1.4980 - acc: 0.5115 - val_loss: 1.4021 - val_acc: 0.5472
Epoch 11/30
 - 16s - loss: 1.3666 - acc: 0.5673 - val_loss: 1.2505 - val_acc: 0.6004
Epoch 12/30
 - 16s - loss: 1.2874 - acc: 0.5929 - val_loss: 1.2535 - val_acc: 0.6136
Epoch 13/30
 - 16s - loss: 1.1575 - acc: 0.6334 - val_loss: 1.5258 - val_acc: 0.5783
Epoch 14/30
 - 16s - loss: 1.0611 - acc: 0.6654 - val_loss: 1.4557 - val_acc: 0.5475
Epoch 15/30
 - 16s - loss: 0.9924 - acc: 0.6912 - val_loss: 1.0096 - val_acc: 0.6994
Epoch 16/30
 - 16s - loss: 0.8847 - acc: 0.7251 - val_loss: 0.9268 - val_acc: 0.7241
Epoch 17/30
 - 16s - loss: 0.8236 - acc: 0.7462 - val_loss: 0.8659 - val_acc: 0.7323
Epoch 18/30
 - 16s - loss: 0.7279 - acc: 0.7784 - val_loss: 1.2581 - val_acc: 0.6646
Epoch 19/30
 - 16s - loss: 0.6768 - acc: 0.7983 - val_loss: 0.7688 - val_acc: 0.7802
Epoch 20/30
 - 16s - loss: 0.6309 - acc: 0.8124 - val_loss: 0.7780 - val_acc: 0.7713
Epoch 21/30
 - 16s - loss: 0.5905 - acc: 0.8246 - val_loss: 0.6931 - val_acc: 0.8068
Epoch 22/30
 - 16s - loss: 0.5342 - acc: 0.8437 - val_loss: 0.7131 - val_acc: 0.7978
Epoch 23/30
 - 16s - loss: 0.4710 - acc: 0.8647 - val_loss: 0.7079 - val_acc: 0.8013
Epoch 24/30
 - 16s - loss: 0.5082 - acc: 0.8526 - val_loss: 0.7009 - val_acc: 0.8073
Epoch 25/30
 - 16s - loss: 0.4109 - acc: 0.8825 - val_loss: 0.6530 - val_acc: 0.8310
Epoch 26/30
```

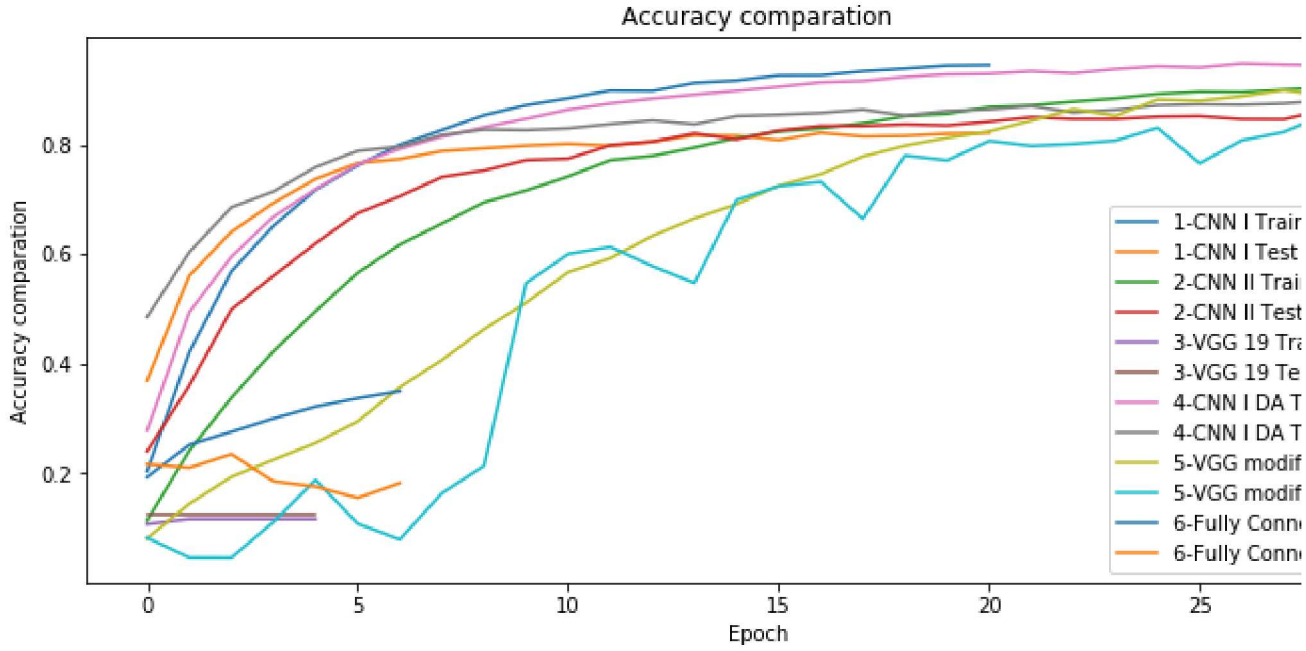
```
 - 16s - loss: 0.4238 - acc: 0.8804 - val_loss: 0.8536 - val_acc: 0.7660
Epoch 27/30
 - 16s - loss: 0.3983 - acc: 0.8884 - val_loss: 0.7216 - val_acc: 0.8078
Epoch 28/30
 - 16s - loss: 0.3518 - acc: 0.8992 - val_loss: 0.7410 - val_acc: 0.8234
Epoch 29/30
 - 16s - loss: 0.3952 - acc: 0.8891 - val_loss: 0.5520 - val_acc: 0.8560
Epoch 30/30
 - 16s - loss: 0.3385 - acc: 0.9060 - val_loss: 0.5719 - val_acc: 0.8560

Model 6 -------------
Train on 15193 samples, validate on 3799 samples
Epoch 1/30
 - 3s - loss: 2.6525 - acc: 0.1934 - val_loss: 2.5529 - val_acc: 0.2169
Epoch 2/30
 - 2s - loss: 2.4389 - acc: 0.2522 - val_loss: 2.5706 - val_acc: 0.2103
Epoch 3/30
 - 2s - loss: 2.3484 - acc: 0.2760 - val_loss: 2.5035 - val_acc: 0.2351
Epoch 4/30
 - 2s - loss: 2.2739 - acc: 0.3003 - val_loss: 2.7079 - val_acc: 0.1853
Epoch 5/30
 - 2s - loss: 2.2202 - acc: 0.3218 - val_loss: 2.6196 - val_acc: 0.1758
Epoch 6/30
 - 2s - loss: 2.1653 - acc: 0.3373 - val_loss: 2.7500 - val_acc: 0.1553
Epoch 7/30
 - 2s - loss: 2.1194 - acc: 0.3502 - val_loss: 2.7800 - val_acc: 0.1822
```

Resultados

```
def plot_compare_alb(vhistory, vname, metric_train, metric_val, title="Graph title"):
  plt.figure(figsize=(12,5))
  for i in range (len(vhistory)):

    plt.plot(vhistory[i].history[metric_train],label='%s Train' % vname[i])

    plt.plot(vhistory[i].history[metric_val],label='%s Test' % vname[i])
  #plt.figure(figsize=(4,5))
  #plt.rcParams["figure.figsize"] = [10,10]
  plt.title(title)
  plt.ylabel(title)
  plt.xlabel('Epoch')
  plt.legend(loc='lower right')
  plt.show()

def plot_compare_alb_loss_acc(vhistory, vname):
  plot_compare_alb(vhistory, vname, 'loss','val_loss', title="Loss comparation")
  plot_compare_alb(vhistory, vname,'acc', 'val_acc', title="Accuracy comparation")




#graphs
vhistory = [history1,history2,history3,history4,history5,history6]
vname = ['1-CNN I','2-CNN II','3-VGG 19','4-CNN I DA','5-VGG modified','6-Fully Connected'
```

```
plot_compare_all_loss_acc(vhistory, vname)
```





Tras analizar varios modelos hemos llegado a las siguientes

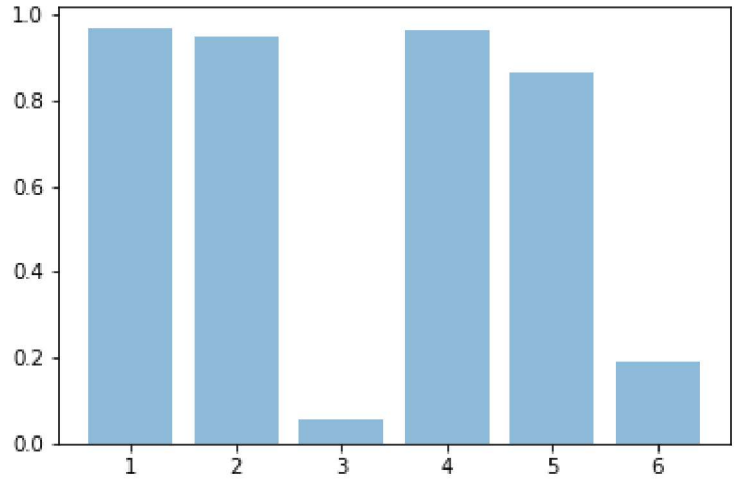
- El tiempo de entrenamiento es un factor a tener en cuenta, debido a un early stopping que h dar cuenta de la precision final de cada modelo. Sin embargo si lo que nos interesa es tener buenos resultados CNN-I destaca respecto al resto ya que en 8 epochs tenemos un modelo complejos como VGG19 ha tardado mas en obtener buenos resultados pero ha acabo por s tiempo de entrenamiento y ajuste.
- El modelo fully conected (modelo6) no ofrece muy buenos resultados.
- Los modelos de CNN son mas precisos que un FCL para este tipo de problemas en los que
- Al introducir Data Augmentation hemos disminuido levenmente la precision del modelo, per validacion y entrenamiento convergen, lo que significa que el modelo ha aprendido.

- El modelo preentrenado no resulta para nada eficaz para este dataset en concreto.
- Utilizando transfer learning (modelo5) hemos podido mejorar la precision pero hace falta m precisiones similares a las logradas con los modelos 1 y 2.
- El mejor modelo es el modelo 1 y cuando lo aplicamos con Data Augmentation.

Test Images

```
models = [model1,model2,model3,model4,model5,model6]
a = 0
x = list()
y = list()

for i in models:
  a += 1
  if (a>5):
    score = i.evaluate(xtestfc,ytestfc,verbose=0)
  else:
    score = i.evaluate(xtest,ytest,verbose=0)
  print('Model {} : '.format(a))
  print('Test loss = ',score[0])
  print('Test accuracy = ', score[1])
  x.append(a)
  y.append(score[1])
  print()

# grafica
plt.bar(np.arange(len(y)),y, align='center', alpha=0.5)
plt.xticks(np.arange(len(y)), x);
plt.show()
```

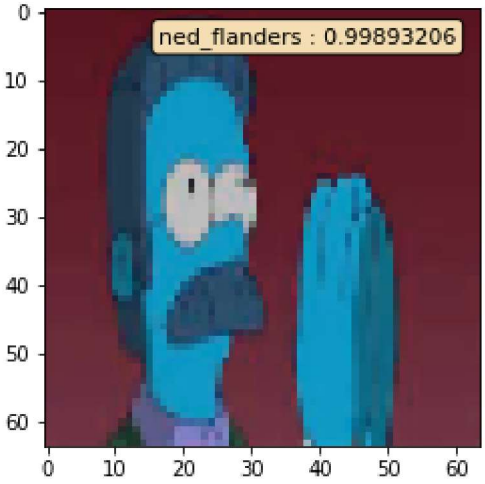
```
Model 1 :
Test loss =  0.13145169507753984
Test accuracy =  0.9685393251729815

Model 2 :
Test loss =  0.20055343756514987
Test accuracy =  0.947191011369898

Model 3 :
Test loss =  2.979243161705103
Test accuracy =  0.05617977538135614

Model 4 :
Test loss =  0.15734580542599216
Test accuracy =  0.9629213484485498

Model 5 :
Test loss =  0.42009740304411125
Test accuracy =  0.8629213477788347

Model 6 :
Test loss =  2.7031880009040403
Test accuracy =  0.1887640450275346
```

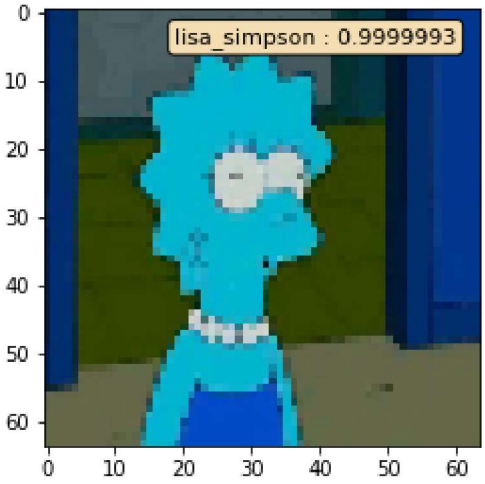


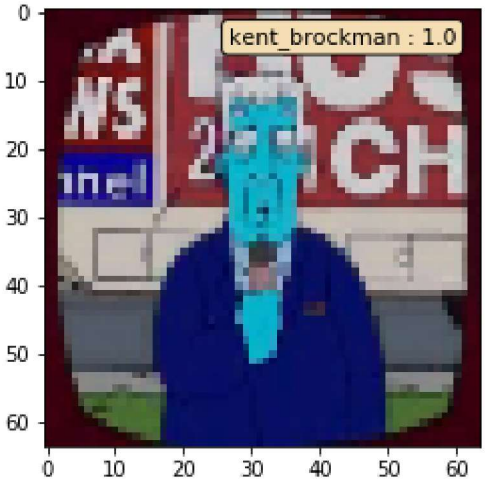
El modelo 4 - CNN ajustada propia con data augmentation es la que ofrece mejores resultados y t

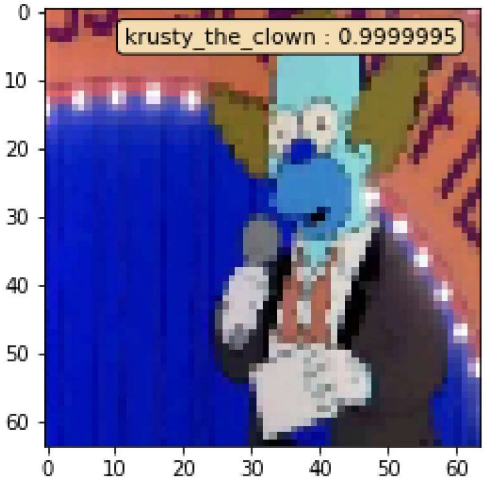
```
def visualize_example(x,s):
    plt.figure()
    plt.imshow(x)
    #plt.colorbar()
    plt.grid(False)
    props = dict(boxstyle='round', facecolor='wheat', alpha=1)
    plt.text(60, 2, s, size=11, rotation=0,ha="right", va="top",bbox=dict(props))
    plt.show()


mlist = np.random.randint(0, len(xtest), size=6)

for num in mlist:
  # hacemos una prediccion
```

```
mypred = model4.predict(np.array([xtest[num]]))
nclass = np.argmax(mypred)
pred =  list(MAP_CHARACTERS.values())[nclass] #prediccion
#print('Class == ',str(pred))

s = str(pred)+' : ' + str(np.max(mypred))
# dibujamos
visualize_example(xtest[num],s)
```

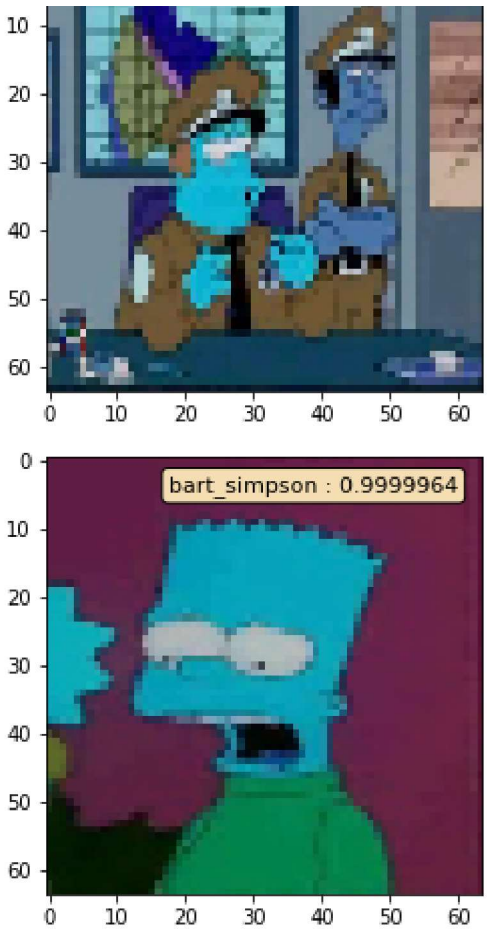

ned_flanders : 0.99893206


lisa_simpson : 0.9999993


kent_brockman : 1.0


krusty_the_clown : 0.9999995


chief_wiggum : 0.99998605



Vemos como con el modelo4 obtenemos muy buenas precisiones en el test.

Conclusiones finales

Este dataset, al ser mas complejo que el fashion MNIST ha sido mas difícil entrenarlo. Sin embar
resultados. Las conclusiones generales del lqborqtiorio son los siguientes:

- Hay muchas posibilidades a la hora de elegir un modelo de CNN. En cuanto a dimensiones,
introducido varias capas convolutivas de cada vez menor tamaño (en torno a la mitad a cad
- Los modelos CNN en promedio convergen más rápido que los Fully Connected y utilizan ba
- Utilizar Data Augmentation ha reducido considerablemente el overfiting.
- El tiempo disponible para realizar nuestro entrenamiento es otro factor a tener en cuenta y h
Podemos obtener una buena precisión también con una fully conected layer, pero nos ha lle
relajando las condiciones de early stoping.
- Utilizar transfer learning es buena idea si queremos un modelo rápido con poca precisión o
reajustando los parámetros por defecto.

Guardar en pdf

1 celda oculta