Facultad de Ciencias, UNAM
Redes Neuronales
Tarea 5

Rubí Rojas Tania Michelle

1 de junio de 2020

1. Construye una red neuronal convolucional para detectar las imágenes en el *dataset* de perros y gatos.

   SOLUCIÓN: Primero, descomprimimos cada uno de nuestros archivos *.zip* y los almacenamos en una carpeta llamada *dataset*:

```
from zipfile import ZipFile

# Le hacemos un unzip a nuestro conjunto de entrenamiento.
with ZipFile('dataset/train_dogscats.zip', 'r') as zipObj:
    zipObj.extractall('dataset')

# Le hacemos un unzip a nuestro conjunto de prueba.
with ZipFile('dataset/test_dogscats.zip', 'r') as zipObj:
    zipObj.extractall('dataset')

```

   Creamos un *dataframe* con cada una de las imágenes dentro de la carpeta *train*. Como el nombre de cada imagen nos indica si el animalito es un perro o es un gato, extraeremos esa información.

```
import os
import pandas as pd

nombres = os.listdir("dataset/train")
categorias = []
for nombre in nombres:
    animal = nombre.split('.')[0] # Obtenemos el nombre del animalito.
    if(animal == 'dog'):
        categorias.append(1)
    else:
        categorias.append(0)

data_frame = pd.DataFrame({'nombre'   : nombres,
                           'categoria': categorias
                           })

# Visualizamos el dataframe con 10 archivos.
data_frame.head(10)

```

	nombre	categoria
0	cat.3145.jpg	0
1	cat.7.jpg	0
2	dog.3377.jpg	1
3	dog.399.jpg	1
4	cat.2181.jpg	0
5	cat.5778.jpg	0
6	cat.7417.jpg	0
7	dog.11297.jpg	1
8	dog.11391.jpg	1
9	cat.4047.jpg	0

Así, tenemos un *dataframe* con cada una de las imágenes y sus respectivas etiquetas.

Ahora, dividimos nuestro conjunto de datos en entrenamiento y prueba.

```
from sklearn.model_selection import train_test_split

# Convertimos la variable "categoria" en gato o perro.
data_frame["categoria"] = data_frame["categoria"].replace({0:'gato',
                                                           1:'perro'
                                                          })

# Dividimos nuestro conjunto de datos con un split del 75-25 para
# entrenamiento y prueba.
X_train, X_test = train_test_split(data_frame,
                                   test_size    = 0.25,
                                   random_state = 42)

# Reestablecemos los indices de nuestros datos.
X_train = X_train.reset_index(drop = True)
X_test = X_test.reset_index(drop = True)

# Obtenemos su respectivo numero de columnas.
total_train = X_train.shape[0]
total_test = X_test.shape[0]

```

```
from keras.preprocessing.image import ImageDataGenerator, load_img

# Modificamos las imagenes para poder obtener mas datos con los que
# entrenar.
datos_train = ImageDataGenerator(rotation_range     = 15,
                                 rescale            = 1./255, # Normalizamos.
                                 shear_range        = 0.1,
                                 zoom_range         = 0.2,
                                 horizontal_flip    = True,
                                 width_shift_range  = 0.1,
                                 height_shift_range = 0.1
                                )


```

```
15
16    # Generamos las imagenes de entrenamiento.
17    generador_train = datos_train.flow_from_dataframe(X_train,
18                                                      "dataset/train/",
19                                                      x_col       = 'nombre',
20                                                      y_col       = 'categoria',
21                                                      target_size = (128, 128),
22                                                      class_mode  = 'categorical',
23                                                      batch_size  = 15
24                                                     )
25
26    # Generamos las imagenes de prueba.
27    datos_test = ImageDataGenerator(rescale = 1./255)
28    generador_test = datos_test.flow_from_dataframe(X_test,
29                                                    "dataset/train/",
30                                                    x_col       = 'nombre',
31                                                    y_col       = 'categoria',
32                                                    target_size = (128, 128),
33                                                    class_mode  ='categorical',
34                                                    batch_size  = 15
35                                                   )
36
```

```
Found 18750 validated image filenames belonging to 2 classes.
Found 6250 validated image filenames belonging to 2 classes.
```

Luego, diseñamos el modelo de nuestra red convolucional.

```
1     from keras.models import Sequential
2     from keras.layers import Convolution2D, MaxPooling2D, Dropout, Flatten,
3                              Dense, Activation, BatchNormalization
4
5     model = Sequential()
6
7     # Primer capa.
8     model.add(Convolution2D(32, (3, 3), activation = 'relu',
9                             input_shape = (128, 128, 3)))
10    model.add(BatchNormalization())
11    model.add(MaxPooling2D(pool_size = (2, 2)))
12    model.add(Dropout(0.25))
13
14    # Segunda capa.
15    model.add(Convolution2D(64, (3, 3), activation='relu'))
16    model.add(BatchNormalization())
17    model.add(MaxPooling2D(pool_size = (2, 2)))
18    model.add(Dropout(0.25))
19
20    # Tercer capa.
21    model.add(Convolution2D(128, (3, 3), activation = 'relu'))
22    model.add(BatchNormalization())
23    model.add(MaxPooling2D(pool_size = (2, 2)))
24    model.add(Dropout(0.25))
25
26    # Cuarta Capa: fully-connected.
27    model.add(Flatten())
28    model.add(Dense(512, activation='relu'))
29    model.add(BatchNormalization())
30    model.add(Dropout(0.5))
```

```
31
32      # Clasificador Softmax.
33      model.add(Dense(2, activation = 'softmax'))
34
```

Analizamos el modelo.

```
1           model.summary()
2
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 126, 126, 32)      896
_________________________________________________________________
batch_normalization_17 (Batc (None, 126, 126, 32)      128
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 63, 63, 32)        0
_________________________________________________________________
dropout_17 (Dropout)         (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 61, 61, 64)        18496
_________________________________________________________________
batch_normalization_18 (Batc (None, 61, 61, 64)        256
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 30, 30, 64)        0
_________________________________________________________________
dropout_18 (Dropout)         (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 28, 28, 128)       73856
_________________________________________________________________
batch_normalization_19 (Batc (None, 28, 28, 128)       512
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 14, 14, 128)       0
_________________________________________________________________
dropout_19 (Dropout)         (None, 14, 14, 128)       0
_________________________________________________________________
flatten_5 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_9 (Dense)              (None, 512)               12845568
_________________________________________________________________
batch_normalization_20 (Batc (None, 512)               2048
_________________________________________________________________
dropout_20 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_10 (Dense)             (None, 2)                 1026
=================================================================
Total params: 12,942,786
Trainable params: 12,941,314
Non-trainable params: 1,472
```

Finalmente, lo compilamos.

```
1       model.compile(loss      = 'categorical_crossentropy',
2                     optimizer = 'rmsprop',
3                     metrics   = ['accuracy'])
4
```

Ahora, toca el turno del entrenamiento. Notemos que *EarlyStopping* lo utilizamos para evitar el exceso de ajuste (detenemos el aprendizaje después de que *epochs* = 10 y el valor de val_loss no haya disminuido) y *ReduceLROPlateau* lo utilizamos para manejar la tasa de aprendizaje de la red (la reducimos cuando la precisión no aumenta en 2 steps).

```
1       from keras.callbacks import EarlyStopping, ReduceLROnPlateau
2
3       H = model.fit(generador_train,
4                     epochs           = 10,
5                     validation_data  = generador_test,
6                     validation_steps = total_test // 15,
7                     steps_per_epoch  = total_train // 15,
8                     callbacks        = [EarlyStopping(patience = 10),
9                                         ReduceLROnPlateau(monitor  = 'val_accuracy',
10                                                          patience = 2, verbose = 1,
11                                                          factor = 0.5,min_lr = 0.00001)]
12                    )
13
```

```
Epoch 1/10
1250/1250 [==============================] - 1539s 1s/step - loss: 0.7743 - accuracy: 0.6187 - val_loss: 0.5320 - v
al_accuracy: 0.6755
Epoch 2/10
1250/1250 [==============================] - 1556s 1s/step - loss: 0.5628 - accuracy: 0.7194 - val_loss: 0.4087 - v
al_accuracy: 0.7440
Epoch 3/10
1250/1250 [==============================] - 1510s 1s/step - loss: 0.5171 - accuracy: 0.7497 - val_loss: 0.4526 - v
al_accuracy: 0.7904
Epoch 4/10
1250/1250 [==============================] - 1577s 1s/step - loss: 0.4806 - accuracy: 0.7769 - val_loss: 0.4198 - v
al_accuracy: 0.7801
Epoch 5/10
1250/1250 [==============================] - 1323s 1s/step - loss: 0.4558 - accuracy: 0.7891 - val_loss: 0.1946 - v
al_accuracy: 0.8141
Epoch 6/10
1250/1250 [==============================] - 1321s 1s/step - loss: 0.4283 - accuracy: 0.8096 - val_loss: 0.3005 - v
al_accuracy: 0.8279
Epoch 7/10
1250/1250 [==============================] - 1306s 1s/step - loss: 0.4086 - accuracy: 0.8175 - val_loss: 0.8404 - v
al_accuracy: 0.7541
Epoch 8/10
1250/1250 [==============================] - 1319s 1s/step - loss: 0.3992 - accuracy: 0.8228 - val_loss: 0.9680 - v
al_accuracy: 0.6277

Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
Epoch 9/10
1250/1250 [==============================] - 1327s 1s/step - loss: 0.3576 - accuracy: 0.8428 - val_loss: 0.2318 - v
al_accuracy: 0.8574
Epoch 10/10
1250/1250 [==============================] - 1323s 1s/step - loss: 0.3439 - accuracy: 0.8490 - val_loss: 0.6174 - v
al_accuracy: 0.8441
```

Evaluamos qué tan bien le fue con nuestro conjunto de prueba.

```
1  score = model.evaluate(generador_train, steps = len(generador_train))
2
```

```
1250/1250 [==============================] - 330s 264ms/step
```

```
1  print("Score {}".format(score))
2
```

```
Score [0.23800379037857056, 0.8423466682434082]
```

Obtenemos un 0.84 % de precisión.

2. ¿Qué arquitectura diseñaste?

   SOLUCIÓN: Diseñamos una *red neuronal convolucional* con la siguiente estructura

   - Capa Uno

     Tiene una capa convolucional de 3×3 (sin paddings, aprendemos un total de 32 filtros, utilizamos la función de activación $ReLU$ e indicamos el tamaño de las imágenes), una capa de Batch Normalization (para aumentar la estabilidad de la red neuronal), una capa de MaxPooling de 2×2 y una capa de Dropout de 0.25.

   - Capa Dos

     Tiene una capa convolucional de 3×3 (sin paddings, aprendemos un total de 64 filtros y utilizamos la función de activación $ReLU$), una capa de Batch Normalization (para aumentar la estabilidad de la red neuronal), una capa de MaxPooling de 2×2 y una capa de Dropout de 0.25.

   - Capa Tres

     Tiene una capa convolucional de 3×3 (sin paddings, aprendemos un total de 128 filtros y utilizamos la función de activación $ReLU$), una capa de Batch Normalization (para aumentar la estabilidad de la red neuronal), una capa de MaxPooling de 2×2 y una capa de Dropout de 0.25.

   - Cuarta Capa: fully-connected.

     Añadimos una única capa fully-connected con 512 nodos (cuya función de activación es $ReLU$) a nuestra red convolucional.

   - Clasificador Softmax.

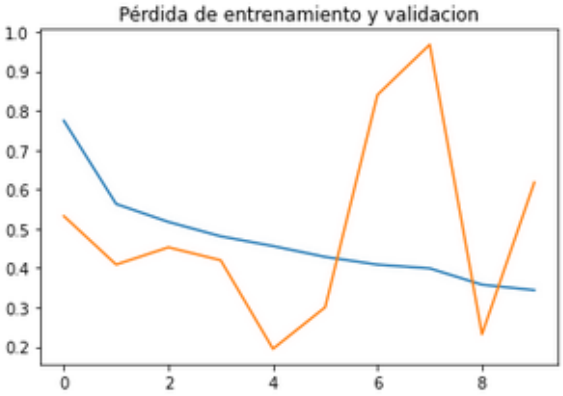
     Añadimos el clasificador Softmax (vamos a tener dos clases: perros y gatos). La salida de esta capa son los valores propios de predicción.

3. Haz las gráficas de pérdida y de precisión tanto para el entrenamiento como para la validación y preséntalas.

   SOLUCIÓN:

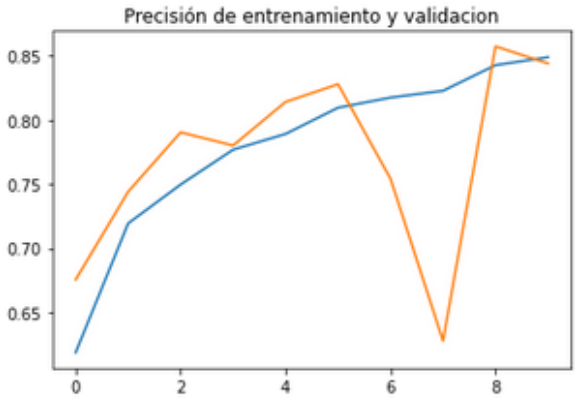
   - Pérdida

```
        import matplotlib.pyplot as plt

        plt.plot(H.history['loss'])
        plt.plot(H.history['val_loss'])
        plt.title('Perdida de entrenamiento y validacion')

```



- Precisión

```
        plt.plot(H.history['accuracy'])
        plt.plot(H.history['val_accuracy'])
        plt.title('Precision de entrenamiento y validacion')

```



4. Salva tu modelo y súbelo.

   SOLUCIÓN: El modelo lo guardamos de la siguiente manera

```
    model.save('mi_modelo.h5')

```

5. Úsalo para reconocer las 4 imágenes anexas.

   SOLUCIÓN: Realizamos el reconocimiento de cada una de las cuatro imágenes solicitadas.

   - *cat1.jpeg*

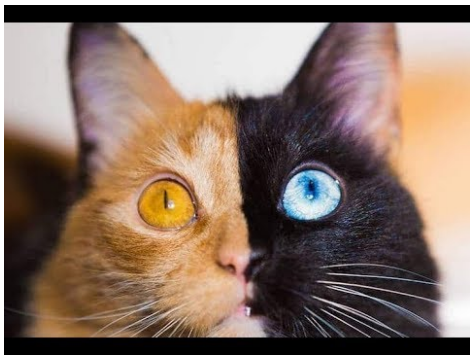
```
        from PIL import Image

        # Definimos los dos resultados posibles.
        categoria_final = {0:'gatito', 1:'perrito'}

        # Cargamos la imagen y la modificamos para poder hacer el
        # reconocimiento.
        imagen1 = Image.open("imagenes/cat1.jpeg")
        imagen1 = imagen1.resize((128, 128))
        imagen1 = np.expand_dims(imagen1, axis = 0)
        imagen1 = np.array(imagen1)
        imagen1 = imagen1 / 255

        # Realizamos la prediccion.
        reconocimiento = model.predict_classes([imagen1])[0]

        # Visualizamos el resultado de la prediccion.
        print(reconocimiento, categoria_final[reconocimiento])

```

```
0 gatito
```

- *cat2.jpg*

```
        # Cargamos la imagen y la modificamos para poder hacer el
        # reconocimiento.
        imagen2 = Image.open("imagenes/cat2.jpg")
        imagen2 = imagen2.resize((128, 128))
        imagen2 = np.expand_dims(imagen2, axis = 0)
        imagen2 = np.array(imagen2)
        imagen2 = imagen2 / 255

        # Realizamos la prediccion.
        reconocimiento = model.predict_classes([imagen2])[0]

        # Visualizamos el resultado de la prediccion.
        print(reconocimiento, categoria_final[reconocimiento])

```

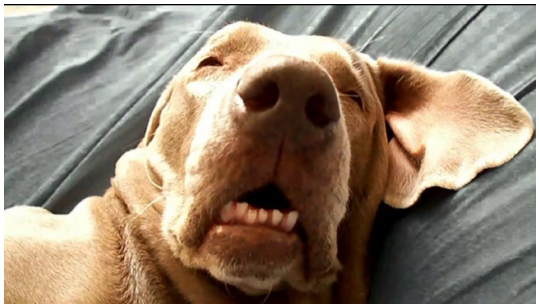
```
0 gatito
```



- *dog1.jpeg*

```
        # Cargamos la imagen y la modificamos para poder hacer el
        # reconocimiento.
        imagen3 = Image.open("imagenes/dog1.jpeg")
        imagen3 = imagen3.resize((128, 128))
        imagen3 = np.expand_dims(imagen3, axis = 0)
        imagen3 = np.array(imagen3)
        imagen3 = imagen3 / 255

        # Realizamos la prediccion.
        reconocimiento = model.predict_classes([imagen3])[0]

        # Visualizamos el resultado de la prediccion.
        print(reconocimiento, categoria_final[reconocimiento])

```

```
1 perrito
```



- *dog2.jpeg*

```
        # Cargamos la imagen y la modificamos para poder hacer el
        # reconocimiento.
        imagen4 = Image.open("imagenes/dog2.jpeg")
        imagen4 = imagen4.resize((128, 128))
        imagen4 = np.expand_dims(imagen4, axis = 0)
        imagen4 = np.array(imagen4)
        imagen4 = imagen4 / 255

        # Realizamos la prediccion.
        reconocimiento = model.predict_classes([imagen4])[0]

        # Visualizamos el resultado de la prediccion.
        print(reconocimiento, categoria_final[reconocimiento])

```

```
1 perrito
```