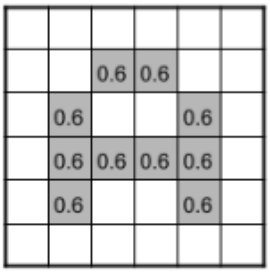
Creado por:

Isabel Maniega

Convolutional Neural Networks (CNN)

Es una clase de ANN pero aplicado para analizar imágenes.

Una imagen...

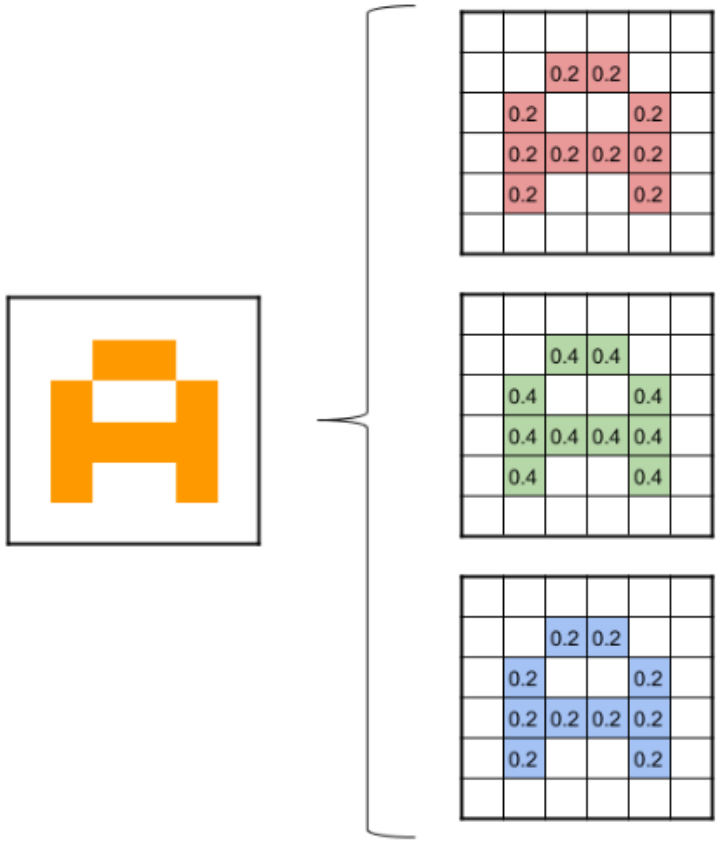


...es una matriz de pixeles.
El valor de los pixeles va de 0 a 255 pero se normaliza para la red neuronal de 0 a 1

Pixeles y neuronas

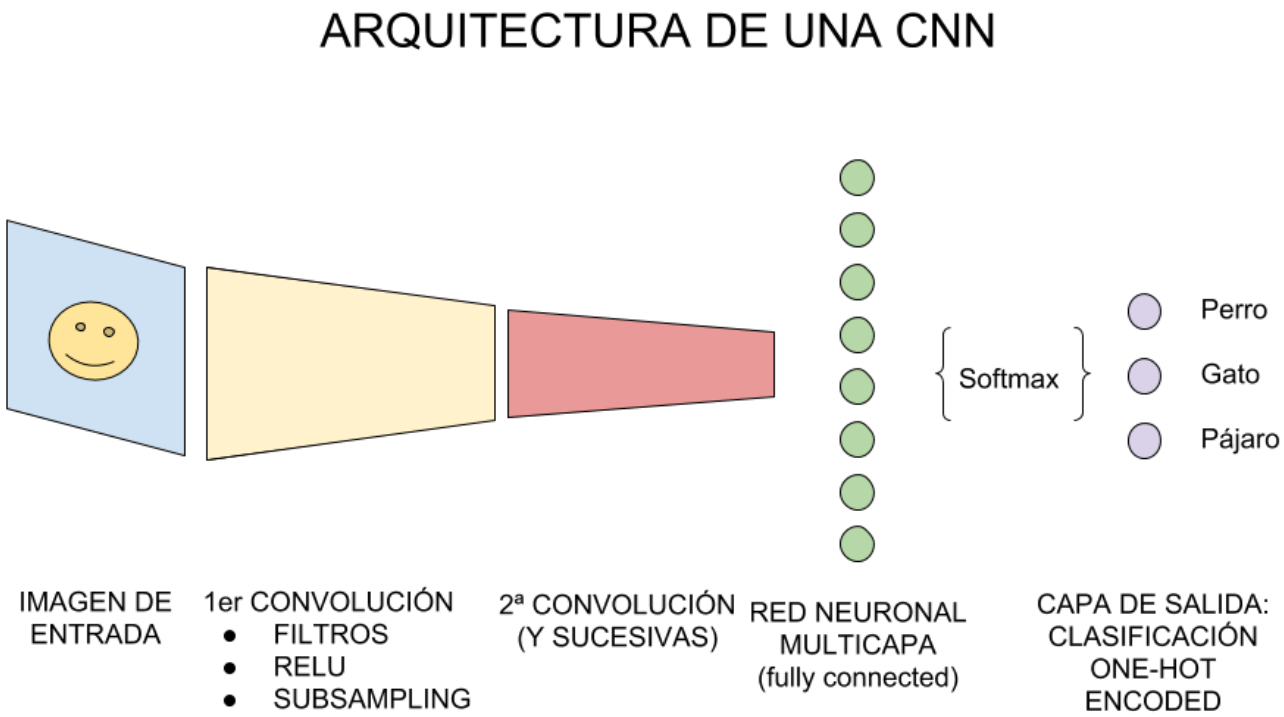
Para comenzar, la red toma como entrada los pixeles de una imagen. Si tenemos una imagen con apenas 28×28 pixeles de alto y ancho, eso equivale a 784 neuronas. Y eso es si sólo tenemos 1 color (escala de grises). Si tuviéramos una imagen a color, necesitaríamos 3 canales (red, green, blue) y entonces usaríamos 28x28x3 = 2352 neuronas de entrada. Esa es nuestra capa de entrada. Para continuar con el ejemplo, supondremos que utilizamos la imagen con 1 sólo color.

Recuerda que como entrada nos conviene normalizar los valores. Los colores de los pixeles tienen valores que van de 0 a 255, haremos una transformación de cada pixel: “valor/255” y nos quedará siempre un valor entre 0 y 1.



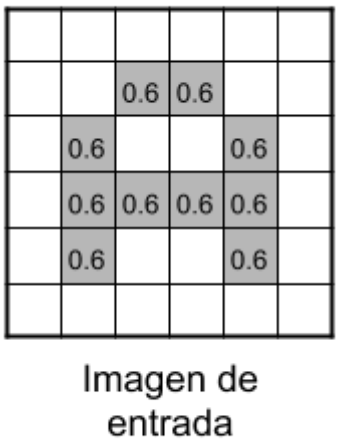
Si la imagen es a color, estará compuesta de tres canales: rojo, verde, azul.

Pasos



Convolution

Estas consisten en tomar “grupos de pixeles cercanos” de la imagen de entrada e ir operando matemáticamente (producto escalar) contra una pequeña matriz que se llama kernel. Ese kernel supongamos de tamaño 3×3 pixels “recorre” todas las neuronas de entrada (de izquierda-derecha, de arriba-abajo) y genera una nueva matriz de salida, que en definitiva será nuestra nueva capa de neuronas ocultas. NOTA: si la imagen fuera a color, el kernel realmente sería de 3x3x3 es decir: un filtro con 3 kernels de 3×3; luego esos 3 filtros se suman (y se le suma una unidad bias) y conformarán 1 salida (cómo si fuera 1 solo canal).

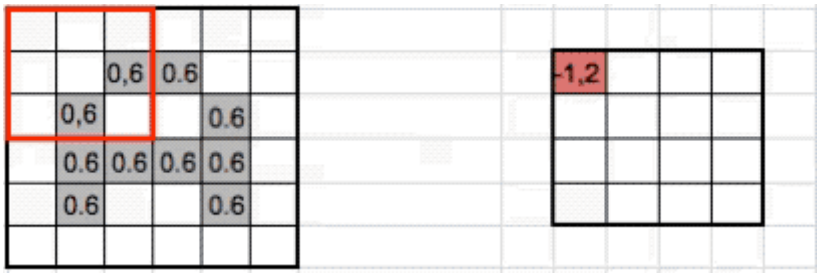


1	0	-1
2	0	-2
1	0	-1

kernel

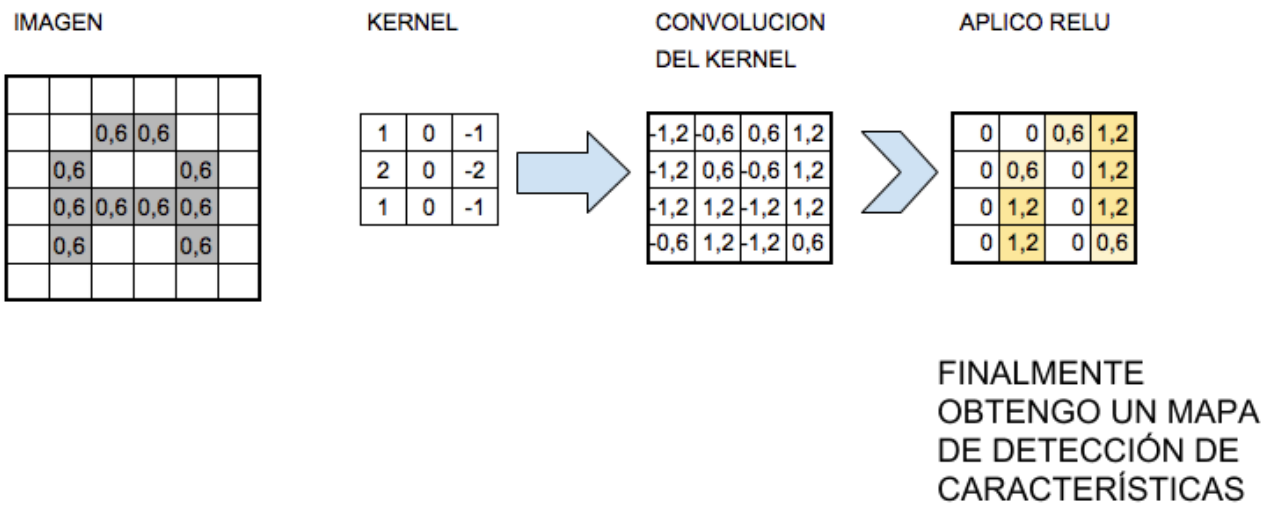
El kernel tomará inicialmente valores aleatorios(1) y se irán ajustando mediante backpropagation. (1)Una mejora es hacer que siga una distribución normal siguiendo simetrías, pero sus valores son aleatorios.

UN DETALLE: en realidad, no aplicaremos 1 sólo kernel, si no que tendremos muchos kernel (su conjunto se llama filtros). Por ejemplo en esta primer convolución podríamos tener 32 filtros, con lo cual realmente obtendremos 32 matrices de salida (este conjunto se conoce como “feature mapping”), cada una de 28x28x1 dando un total del 25.088 neuronas para nuestra PRIMER CAPA OCULTA de neuronas. ¿No les parecen muchas para una imagen cuadrada de apenas 28 pixeles? Imaginen cuántas más serían si tomáramos una imagen de entrada de 224x224x3 (que aún es considerado un tamaño pequeño)…



Aquí vemos al kernel realizando el producto matricial con la imagen de entrada y desplazando de a 1 pixel de izquierda a derecha y de arriba-abajo y va generando una nueva matriz que compone al mapa de features

A medida que vamos desplazando el kernel y vamos obteniendo una “nueva imagen” filtrada por el kernel. En esta primer convolución y siguiendo con el ejemplo anterior, es como si obtuviéramos 32 “imágenes filtradas nuevas”. Estas imágenes nuevas lo que están “dibujando” son ciertas características de la imagen original. Esto ayudará en el futuro a poder distinguir un objeto de otro (por ej. gato ó perro).



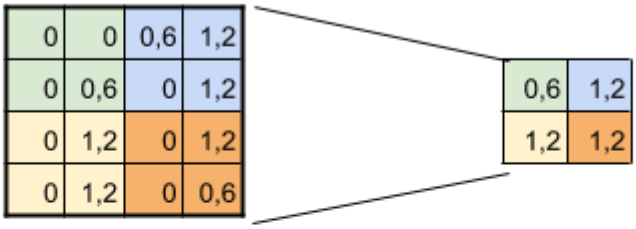
Activación

La función de activación más utilizada para este tipo de redes neuronales es la llamada ReLu por Rectifier Linear Unit y consiste en f(x)=max(0,x).

Ahora viene un paso en el que reduciremos la cantidad de neuronas antes de hacer una nueva convolución. Para reducir el tamaño de la próxima capa de neuronas haremos un proceso de subsampling en el que reduciremos el tamaño de nuestras imágenes filtradas pero en donde deberán prevalecer las características más importantes que detectó cada filtro. Hay diversos tipos de subsampling, el “más usado”: Max-Pooling

Max-Pooling

Vamos a intentar explicarlo con un ejemplo: supongamos que haremos Max-pooling de tamaño 2×2. Esto quiere decir que recorreremos cada una de nuestras 32 imágenes de características obtenidas anteriormente de 28x28px de izquierda-derecha, arriba-abajo PERO en vez de tomar de a 1 pixel, tomaremos de “2×2” (2 de alto por 2 de ancho = 4 pixeles) e iremos preservando el valor “más alto” de entre esos 4 pixeles (por eso lo de “Max”). En este caso, usando 2×2, la imagen resultante es reducida “a la mitad”y quedará de 14×14 pixeles. Luego de este proceso de subsamplig nos quedarán 32 imágenes de 14×14, pasando de haber tenido 25.088 neuronas a 6272, son bastantes menos y -en teoría- siguen almacenando la información más importante para detectar características deseadas.



SUBSAMPLING:
Aplico Max-Pooling de 2x2
y reduzco mi salida a la mitad

La primer convolución es capaz de detectar características primitivas como lineas ó curvas. A medida que hagamos más capas con las convoluciones, los mapas de características serán capaces de reconocer formas más complejas, y el conjunto total de capas de convoluciones podrá “ver”.

Así realizaremos varias convoluciones hasta analizar la imagen, así en este ejemplo empezamos con una imagen de 28x28px e hicimos 3 convoluciones. Si la imagen inicial hubiese sido mayor (de 224x224px) aún hubiéramos podido seguir haciendo convoluciones.

Análisis

In [1]:

```
# Importing the Keras libraries and packages

from tensorflow.keras.layers import Convolution2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten

import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

```
2022-11-22 09:51:19.530096: I tensorflow/core/platform/cpu_feature_gua
rd.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural
Network Library (oneDNN) to use the following CPU instructions in perf
ormance-critical operations:  AVX2 AVX_VNNI FMA
To enable them in other operations, rebuild TensorFlow with the approp
riate compiler flags.
2022-11-22 09:51:19.726599: I tensorflow/core/util/util.cc:169] oneDNN
custom operations are on. You may see slightly different numerical res
ults due to floating-point round-off errors from different computation
orders. To turn them off, set the environment variable `TF_ENABLE_ONED
NN_OPTS=0`.
2022-11-22 09:51:19.766978: W tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:64] Could not load dynamic library 'libcudart.so.11.
0'; dlerror: libcudart.so.11.0: cannot open shared object file: No suc
h file or directory
2022-11-22 09:51:19.767002: I tensorflow/stream_executor/cuda/cudart_s
tub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up
on your machine.
2022-11-22 09:51:19.801075: E tensorflow/stream_executor/cuda/cuda_bla
s.cc:2981] Unable to register cuBLAS factory: Attempting to register f
actory for plugin cuBLAS when one has already been registered
2022-11-22 09:51:20.487532: W tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:64] Could not load dynamic library 'libnvinfer.so.
7'; dlerror: libnvinfer.so.7: cannot open shared object file: No such
file or directory
2022-11-22 09:51:20.487582: W tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:64] Could not load dynamic library 'libnvinfer_plugi
n.so.7'; dlerror: libnvinfer_plugin.so.7: cannot open shared object fi
le: No such file or directory
2022-11-22 09:51:20.487585: W tensorflow/compiler/tf2tensorrt/utils/py
_utils.cc:38] TF-TRT Warning: Cannot dlopen some TensorRT libraries. I
f you would like to use Nvidia GPU with TensorRT, please make sure the
missing libraries mentioned above are installed properly.
```

In [2]:

```
# Initialising the CNN
classifier = Sequential()

# Step 1 - Convolution
classifier.add(Convolution2D(32, 3, 3, input_shape = (64, 64, 3), activation = 'rel

# Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Adding a second convolutional layer
classifier.add(Convolution2D(32, 3, 3, activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full connection
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))

# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['ac
```

```
2022-11-22 09:53:03.998349: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:980] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-11-22 09:53:03.998483: W tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:64] Could not load dynamic library 'libcudart.so.11.
0'; dlerror: libcudart.so.11.0: cannot open shared object file: No suc
h file or directory
2022-11-22 09:53:03.998516: W tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:64] Could not load dynamic library 'libcublas.so.1
1'; dlerror: libcublas.so.11: cannot open shared object file: No such
file or directory
2022-11-22 09:53:03.998537: W tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:64] Could not load dynamic library 'libcublasLt.so.1
1'; dlerror: libcublasLt.so.11: cannot open shared object file: No suc
h file or directory
2022-11-22 09:53:03.998557: W tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:64] Could not load dynamic library 'libcufft.so.10';
dlerror: libcufft.so.10: cannot open shared object file: No such file
or directory
2022-11-22 09:53:03.998576: W tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:64] Could not load dynamic library 'libcurand.so.1
0'; dlerror: libcurand.so.10: cannot open shared object file: No such
file or directory
2022-11-22 09:53:03.998596: W tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:64] Could not load dynamic library 'libcusolver.so.1
1'; dlerror: libcusolver.so.11: cannot open shared object file: No suc
h file or directory
2022-11-22 09:53:03.998614: W tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:64] Could not load dynamic library 'libcusparse.so.1
1'; dlerror: libcusparse.so.11: cannot open shared object file: No suc
h file or directory
2022-11-22 09:53:03.998632: W tensorflow/stream_executor/platform/defa
ult/dso_loader.cc:64] Could not load dynamic library 'libcudnn.so.8';
dlerror: libcudnn.so.8: cannot open shared object file: No such file o
```

```
r directory
2022-11-22 09:53:03.998640: W tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1934] Cannot dlopen some GPU libraries. Please make sure the
missing libraries mentioned above are installed properly if you would
like to use GPU. Follow the guide at https://www.tensorflow.org/instal
l/gpu (https://www.tensorflow.org/install/gpu) for how to download and
setup the required libraries for your platform.
Skipping registering GPU devices...
2022-11-22 09:53:03.999326: I tensorflow/core/platform/cpu_feature_gua
rd.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural
Network Library (oneDNN) to use the following CPU instructions in perf
ormance-critical operations:  AVX2 AVX_VNNI FMA
To enable them in other operations, rebuild TensorFlow with the approp
riate compiler flags.
```

In [3]:

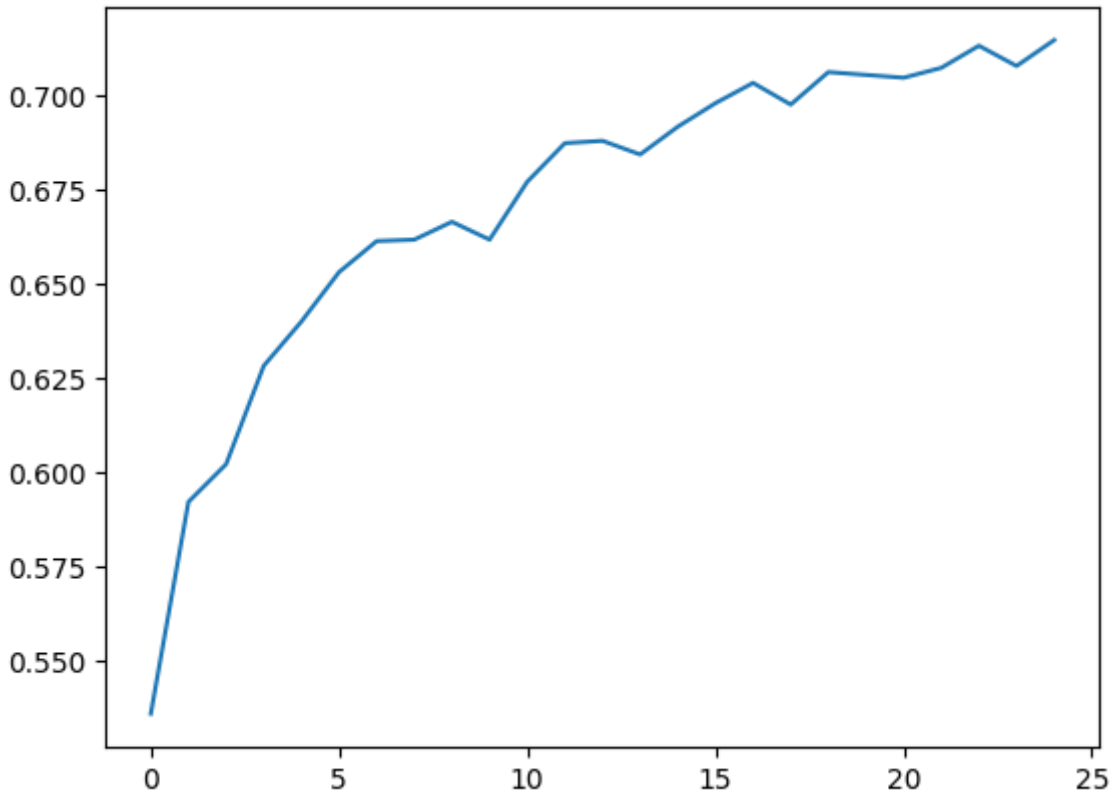
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')

test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')

history = classifier.fit(x=training_set, batch_size = 8000, epochs = 25,
                         validation_data = test_set, validation_batch_size = 2000)
```

```
Found 8000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
Epoch 1/25
250/250 [==============================] - 11s 44ms/step - loss: 0.691
0 - accuracy: 0.5357 - val_loss: 0.6849 - val_accuracy: 0.5700
Epoch 2/25
250/250 [==============================] - 9s 37ms/step - loss: 0.6725
- accuracy: 0.5920 - val_loss: 0.6774 - val_accuracy: 0.5805
Epoch 3/25
250/250 [==============================] - 9s 37ms/step - loss: 0.6583
- accuracy: 0.6020 - val_loss: 0.6478 - val_accuracy: 0.6185
Epoch 4/25
250/250 [==============================] - 9s 37ms/step - loss: 0.6408
- accuracy: 0.6281 - val_loss: 0.6448 - val_accuracy: 0.6205
Epoch 5/25
250/250 [==============================] - 9s 37ms/step - loss: 0.6311
- accuracy: 0.6399 - val_loss: 0.6301 - val_accuracy: 0.6420
Epoch 6/25
250/250 [==============================] - 9s 37ms/step - loss: 0.6206
- accuracy: 0.6530 - val_loss: 0.6430 - val_accuracy: 0.6250
Epoch 7/25
250/250 [==============================] - 10s 39ms/step - loss: 0.618
0 - accuracy: 0.6612 - val_loss: 0.6117 - val_accuracy: 0.6610
Epoch 8/25
250/250 [==============================] - 10s 38ms/step - loss: 0.612
4 - accuracy: 0.6616 - val_loss: 0.6472 - val_accuracy: 0.6315
Epoch 9/25
250/250 [==============================] - 10s 38ms/step - loss: 0.608
9 - accuracy: 0.6664 - val_loss: 0.6220 - val_accuracy: 0.6520
Epoch 10/25
250/250 [==============================] - 9s 38ms/step - loss: 0.6066
- accuracy: 0.6616 - val_loss: 0.6114 - val_accuracy: 0.6695
Epoch 11/25
250/250 [==============================] - 10s 38ms/step - loss: 0.601
1 - accuracy: 0.6770 - val_loss: 0.6239 - val_accuracy: 0.6560
Epoch 12/25
250/250 [==============================] - 10s 38ms/step - loss: 0.588
```

```
9 - accuracy: 0.6873 - val_loss: 0.6260 - val_accuracy: 0.6535
Epoch 13/25
250/250 [==============================] - 10s 38ms/step - loss: 0.587
5 - accuracy: 0.6879 - val_loss: 0.5964 - val_accuracy: 0.6755
Epoch 14/25
250/250 [==============================] - 10s 39ms/step - loss: 0.587
3 - accuracy: 0.6842 - val_loss: 0.6208 - val_accuracy: 0.6560
Epoch 15/25
250/250 [==============================] - 10s 39ms/step - loss: 0.583
1 - accuracy: 0.6916 - val_loss: 0.6018 - val_accuracy: 0.6805
Epoch 16/25
250/250 [==============================] - 10s 38ms/step - loss: 0.576
6 - accuracy: 0.6979 - val_loss: 0.5920 - val_accuracy: 0.6780
Epoch 17/25
250/250 [==============================] - 10s 38ms/step - loss: 0.572
8 - accuracy: 0.7032 - val_loss: 0.5938 - val_accuracy: 0.6870
Epoch 18/25
250/250 [==============================] - 10s 39ms/step - loss: 0.575
3 - accuracy: 0.6975 - val_loss: 0.6032 - val_accuracy: 0.6755
Epoch 19/25
250/250 [==============================] - 9s 38ms/step - loss: 0.5669
- accuracy: 0.7061 - val_loss: 0.6059 - val_accuracy: 0.6765
Epoch 20/25
250/250 [==============================] - 10s 38ms/step - loss: 0.563
5 - accuracy: 0.7054 - val_loss: 0.5815 - val_accuracy: 0.6885
Epoch 21/25
250/250 [==============================] - 9s 38ms/step - loss: 0.5654
- accuracy: 0.7046 - val_loss: 0.6031 - val_accuracy: 0.6770
Epoch 22/25
250/250 [==============================] - 9s 38ms/step - loss: 0.5651
- accuracy: 0.7072 - val_loss: 0.5890 - val_accuracy: 0.6795
Epoch 23/25
250/250 [==============================] - 9s 37ms/step - loss: 0.5595
- accuracy: 0.7131 - val_loss: 0.6129 - val_accuracy: 0.6675
Epoch 24/25
250/250 [==============================] - 9s 38ms/step - loss: 0.5570
- accuracy: 0.7078 - val_loss: 0.5912 - val_accuracy: 0.6940
Epoch 25/25
250/250 [==============================] - 10s 38ms/step - loss: 0.555
4 - accuracy: 0.7146 - val_loss: 0.6107 - val_accuracy: 0.6665
```

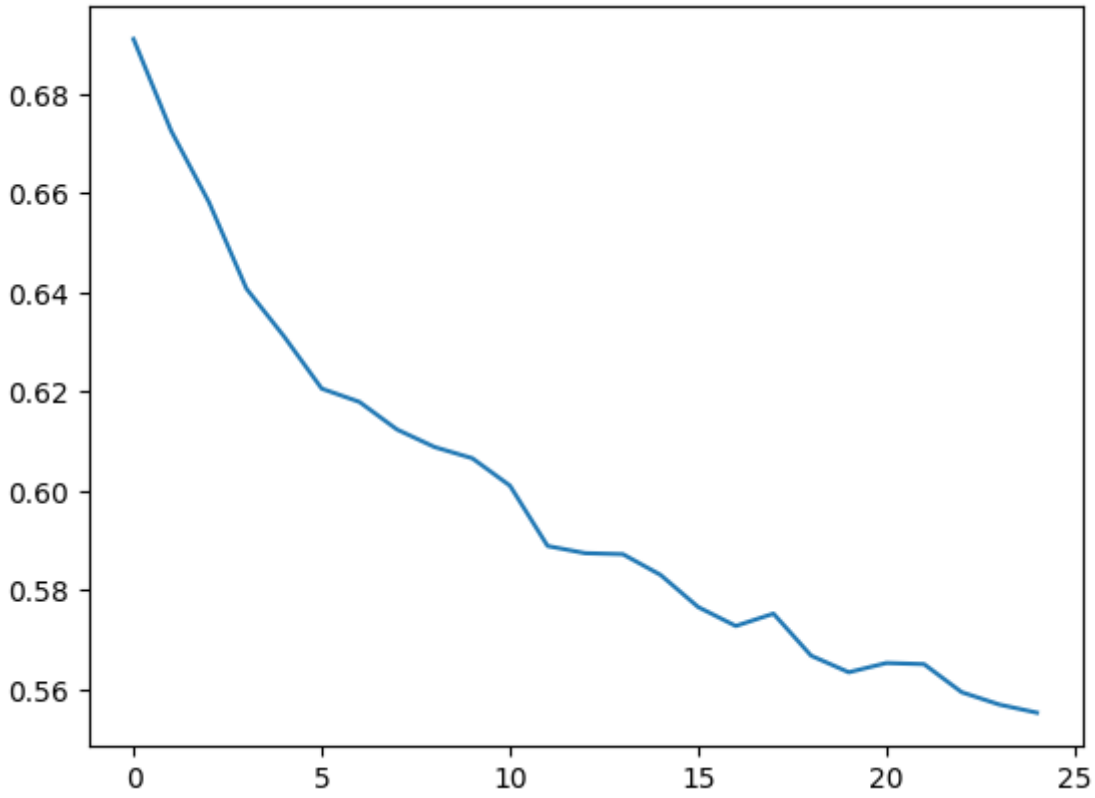
In [4]:

```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'], label = 'train')
plt.show()
```



In [5]:

```
plt.plot(history.history['loss'], label = 'train')
plt.show()
```



In [6]:

```
import numpy as np

#to predict new images
def predict_image(imagepath, classifier):
    predict = tf.keras.preprocessing.image.load_img(imagepath, target_size = (64, 6
    predict_modified = tf.keras.preprocessing.image.img_to_array(predict)
    predict_modified = predict_modified / 255
    predict_modified = np.expand_dims(predict_modified, axis = 0)
    result = classifier.predict(predict_modified)
    if result[0][0] >= 0.5:
        prediction = 'dog'
        probability = result[0][0]
        print ("probability = " + str(probability))
        print("Prediction = " + prediction)
    else:
        prediction = 'cat'
        probability = 1 - result[0][0]
        print ("probability = " + str(probability))
        print("Prediction = " + prediction)
```

In [7]:

```
predict_image("mio.jpg", classifier)
```

```
1/1 [==============================] - 0s 50ms/step
probability = 0.6589929461479187
Prediction = cat
```

In [8]:

```
predict_image("lua.jpeg", classifier)
```

```
1/1 [==============================] - 0s 11ms/step
probability = 0.750931
Prediction = dog
```

In [9]:

```
predict_image("lua_2.jpg", classifier)
```

```
1/1 [==============================] - 0s 12ms/step
probability = 0.8097744
Prediction = dog
```

Creado por:

Isabel Maniega