Sebastián Rodríguez

20003076

Laboratorio # 3

In [28]:
```
import numpy as np
from keras import applications
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.models import Sequential
from keras import backend as K
```

In [29]:
```
# Dimensionar las imágenes que tenemos
img_width, img_height = 150, 150
# Directorios para encontrar los dataset suministrados
Entrena_Dir = 'data/train'
Val_Dir = 'data/validation'

# Cantidad de imagenes utilizadas para entrenamiento y validaciones
# Se coloca una cantidad de epochs en 50 para obtener el coeficiente de 0.8 buscado
Pruebas_ent = 2000
Preubas_Val = 800
epochs = 50
batchsize = 16
```

Red convolucional

Se utilizó una red neuronal convolucional pequeña con pocas capas y pocos filtros por capa, adicional se utilizó aumento y la eliminación de datos.

Abajo es el primer modelo, una simple pila de 3 capas de convolución con una activación ReLU y seguida por capas de máxima acumulación. Algo que llama la atención es el data augmentation puede perturbar las correlaciones aleatorias que pueden estar presentes en el data set sin saberlo.

In [30]:
```
if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

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

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

DATA AUGMENTATION Y PRE-PROCESSING

Vamos a utilizar ImageDataGenerator para aumentar la cantidad de información que podemos obtener de nuestro set de entrenamiento. Vamos a realizar transformaciones en las imágenes que nos ayudará a que el modelo generalice mejor y evitemos overfitting.

In [34]:
```
data_Ent = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

gen_entrenamiento = data_Ent.flow_from_directory(
    Entrena_Dir,
    target_size=(img_width, img_height),
    batch_size=batchsize,
    class_mode='binary')

gen_validacion = test_datagen.flow_from_directory(
    Val_Dir,
    target_size=(img_width, img_height),
    batch_size=batchsize,
    class_mode='binary')

model.fit_generator(
    gen_entrenamiento,
    steps_per_epoch=Pruebas_ent // batchsize,
    epochs=epochs,
    validation_data=gen_validacion,
    validation_steps=Preubas_Val // batchsize)

model.save_weights('Pesos_futuroentrenamiento.h5')
```

```
Found 8000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
Epoch 1/50
125/125 [==============================] - 30s 238ms/step - loss: 0.6199 - accuracy: 0.6
690 - val_loss: 0.5621 - val_accuracy: 0.7050
Epoch 2/50
125/125 [==============================] - 29s 228ms/step - loss: 0.6110 - accuracy: 0.6
765 - val_loss: 0.5720 - val_accuracy: 0.6913
Epoch 3/50
125/125 [==============================] - 30s 239ms/step - loss: 0.6076 - accuracy: 0.6
730 - val_loss: 0.5649 - val_accuracy: 0.7237
Epoch 4/50
125/125 [==============================] - 29s 233ms/step - loss: 0.5976 - accuracy: 0.6
815 - val_loss: 0.7197 - val_accuracy: 0.6750
Epoch 5/50
125/125 [==============================] - 29s 229ms/step - loss: 0.5997 - accuracy: 0.6
795 - val_loss: 0.5034 - val_accuracy: 0.7175
Epoch 6/50
125/125 [==============================] - 29s 230ms/step - loss: 0.5739 - accuracy: 0.7
```

Resultados

Podemos observar que contamos con un accuracy para la validación mientras aumentan los epoch del 0.82 para algunos casos y un 0.69 al inicio de la operación del modelo. En general estamos entre el 0.77 - 0.82 para la exactitud del modelo.

Podríamos utilizar todas las validaciones y usar un modelo que utilice validación cruzada o bien un auto set de pesos y pipelines para mejorar al 0.9%. Otra estrategia sería utilizar más data augementation y variaciones de pesos con un tuning mayo para aumentar la regularización de nuestro sistema.

El modelo conserva los pesos generados de los epoch y batch size que utilizamos como hiper parámetros con el fin de utilizarlos más adelante en un auto tuning más poderoso.

Este modelo es el utilizado actualmente en donde se labora como un identificador de producto que corre en la línea con una exactitud del 0.92 para todos los productos que se consideran como hamburguesa. (Obviamente los data set son distintos).

In [ ]: