PNEUMONIA
IMAGE CLASSIFICATION
with neural network

Domenico Armillotta

INTRODUCTION

The goal is to create an application that classifies radiologies in order to understand whether the patient has pneumonia.
The limits will be those offered by Google colab, i.e. 16gb Ram and limited TPU time, if the results are interesting we will go deeper, creating a professional application, with GUI.

All techniques will be used to try to minimise problems such as overfitting.

This is just an experiment to find a possible field of application in the medical field of neural networks for image classification.
In the event that the preliminary analyses are satisfied, we will proceed to further study and create a usable model.

DATASET

Taken from kaggle :
[Chest X-Ray Images (Pneumonia) | Kaggle]

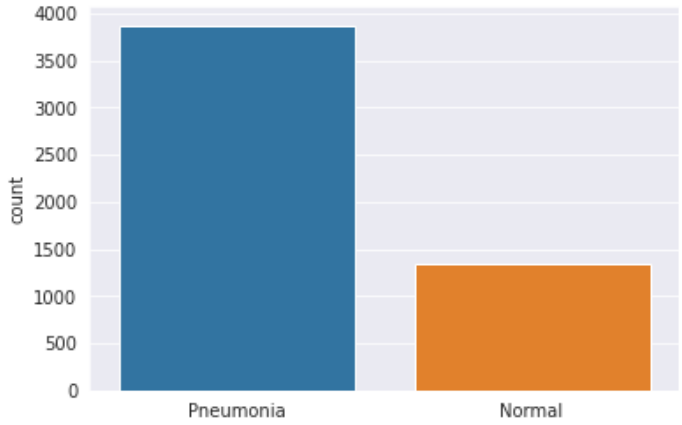
Divided into directories according to class :

- Train - Test - Validation

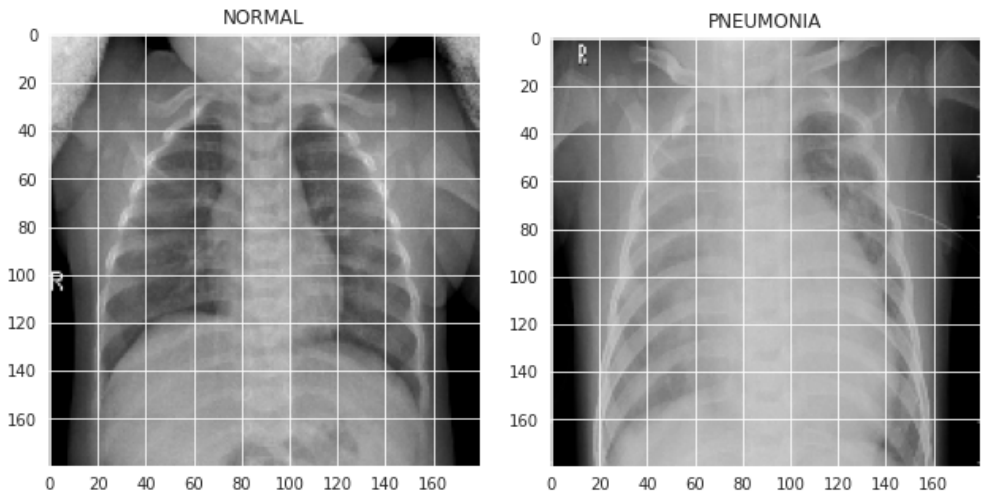
each folder contains :

- Pneumonia - Normal

The training set resulted unbalanced :



Example of data :



DATA PREPARATION

A reshape of the images to 180x180 was done, so that they would fit into Google colab's Ram memory.

Three RGB channels were used, so that the classifier could extrapolate more data.

The data i.e. the images and labels were converted into numpy arrays, so as to give them to the network.
The labels are in one-hot format.

CNN FROM SCRATCH

Having analysed the dataset, we realised that we have very few images, so there is a very high risk of overfitting our networks, so we try to use all possible precautions to avoid this.

PRECAUTION USED :

- multiple **Dropout layers** were used in order to decrease overfitting
- **Batch normalisation layer** was used in order to decrease overfitting and velocity the training
- An attempt was made to reduce the number of parameters that could be driven as far as possible, so as not to overfit the model.
- **early stopping** was used to decrease the risk of overfitting
- **class_weight** was created to manage the unbalanced training set

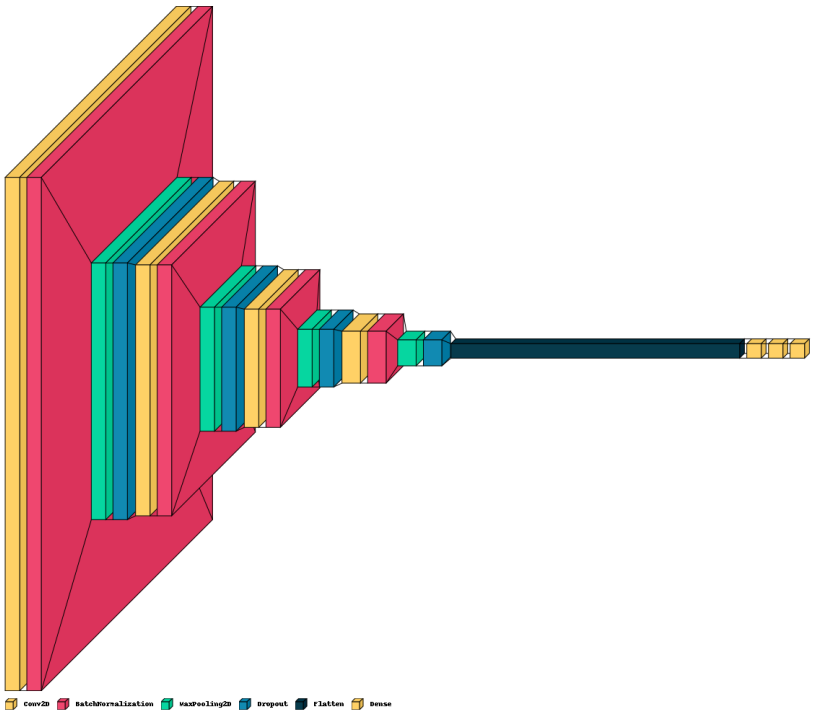
```
#Early stopping
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_prc',
    verbose=1,
    patience=10,
    mode='max',
    restore_best_weights=True)
```

```
#Augmentation
aug = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    vertical_flip=False,
    fill_mode="nearest")
```

```
from sklearn.utils import class_weight

class_weights = class_weight.compute_class_weight(class_weight = "balanced",
                                                  classes = np.unique(y_train),
                                                  y = y_train)

# the field class weight in model.fit() is a dictionary, so we convert class_weights into a dictionary
class_weights = {i : class_weights[i] for i in range(len(class_weights))}
```

ALEXNET MODEL LIKE :

Again, the network overfitted, so the model is unusable for real, robust applications , even if we obtain an accuracy = 85%
Many tests were carried out by changing parameters and values and layers, with little improvement.
This could be caused by the very small training set.
Model used :



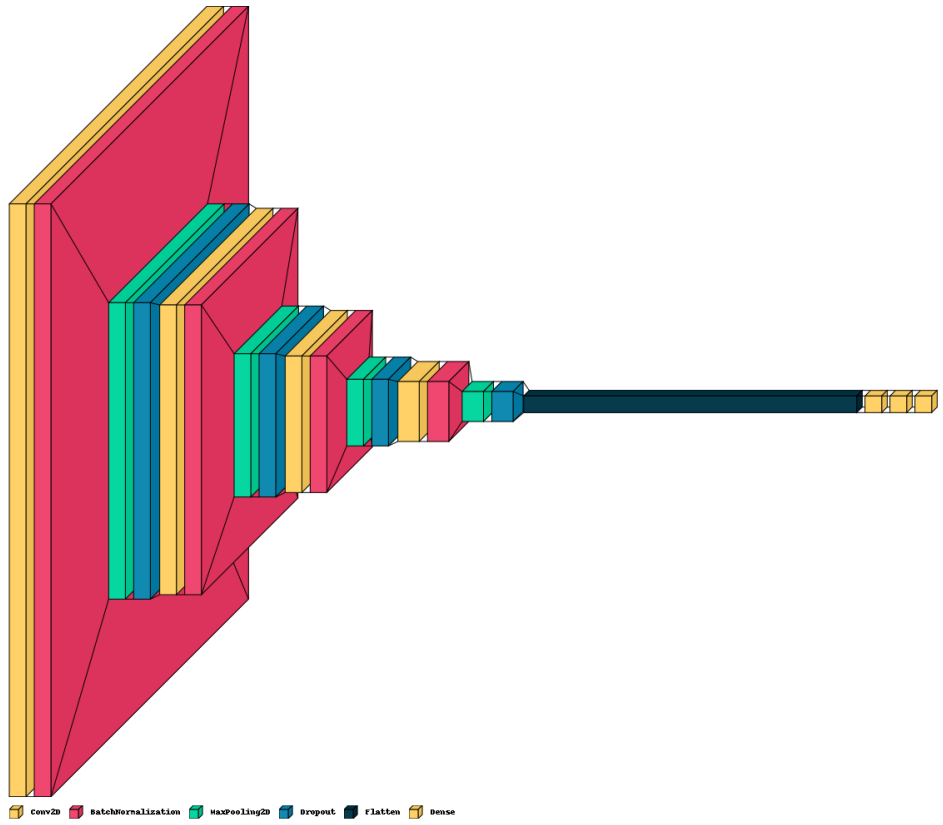
Code :

```
#1st Convolutional Layer
model.add(keras.layers.Conv2D(filters=32,kernel_size=3,activation='relu', input_shape = [img_size, img_size,3]))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=2))
model.add(keras.layers.Dropout(0.2))
#2nd Convolutional Layer
model.add(keras.layers.Conv2D(filters=64,kernel_size=3,activation='relu'))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=2))
model.add(keras.layers.Dropout(0.2))
#3rd Convolutional Layer
model.add(keras.layers.Conv2D(filters=128,kernel_size=3,activation='relu'))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=2))
model.add(keras.layers.Dropout(0.2))
#4th Convolutional Layer
model.add(keras.layers.Conv2D(filters=256,kernel_size=3,activation='relu'))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=2))
model.add(keras.layers.Dropout(0.2))
#Passing it to a Fully Connected layer
model.add(keras.layers.Flatten())

# 1st Fully Connected Layer
model.add(keras.layers.Dense(64,activation='relu'))

# 2st Fully Connected Layer
model.add(keras.layers.Dense(64,activation='relu'))

#Softmax layer for output
model.add(keras.layers.Dense(2,activation='softmax'))
```

MNIST MODEL LIKE

As in alexnet, we have a similar situation here.

Model used :



Code :

```
model = keras.models.Sequential()
# 1 layer
model.add(keras.layers.Conv2D(filters=32,kernel_size=3,activation='relu', input_shape = [img_size, img_size,3]))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=(2,2)))
model.add(keras.layers.Dropout(0.15))
# 2 layer
model.add(keras.layers.Conv2D(filters=64 , activation='relu',kernel_size=3))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=(2,2)))
model.add(keras.layers.Dropout(0.15))
# 3 layer
model.add(keras.layers.Conv2D(filters=128 , activation='relu' ,kernel_size=3))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=(2,2)))
model.add(keras.layers.Dropout(0.15))

model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(64,activation='relu'))
model.add(keras.layers.Dense(2,activation='softmax'))
model.summary()
```

PRE TRAINED CNN - EFFICENTNET

The feature reuse technique was used, i.e. freezing all layers of the efincent net, and attaching a classifier at the end.
The network failed to perform well in terms of accuracy, which was around 37%.
All the precautions mentioned above were used.

```
20/20 [==============================] - 4s 49ms/step
Test Data accuracy:  37.5
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	390
1	0.38	1.00	0.55	234
accuracy			0.38	624
macro avg	0.19	0.50	0.27	624
weighted avg	0.14	0.38	0.20	624

FINAL CONSIDERATION

Since the objective was to create a functioning classifier, the intermediate results were not sufficient to continue the test.
The next steps were to be :
1. explainability of the network
2. examble method on cnn
3. Creation of the application with the GUI

The main cause was overfitting, caused by too small a number of samples, and therefore unusable.

Even the performance of similar projects did not post too much better results, projects on the same dataset from kaggle code and the official keras site were compared.

So we conclude that there is no point in wasting time and resources (time on google colab) on a dataset that would not give us a model for a real-world usable app that is stable in its use.