Final Assignment - Deep Learning and Reinforcement Learning - IBM Machine Learning Professional Certificate

Main Objective

During the latest period many hospital services has been near to collapse due to the pandemic issues. The COVID-patients has abruptly absorbed a large number of resources that was previously dedicated to other patients before the pandemic. Since it is not always able to increase the health care resources due to the large economic effort needed, it is crucial to find an easy and a **cheap solution that could help the health care system to reduce evaluation time and therefore the cost**.

In this sense, **image recognition** is a powerful tool that can help doctors to reduce the decision time and the accuracy of the diagnosis. **Training a model with images previously labelled** a tool can be developed to help the doctors with the diagnosis of certain illness. For this purpose, the **convolutional neural networks** are the best fitted models to solve this problem.

Thus, we were asked by an hospital to develop a model able to **detect pneumonia by looking at a chest x-ray image from a patient**. Our **main goal** then will be to **train a convolutional neural network model to classify chest x-ray images** between two categories, normal (without the illness) and pneumonia.

Brief Description of the Data

The hospital has provided us with a dataset containing **5956** chest x-ray **images** of different patients **labelled** with **pneumonia** or **normal** depending on whether the patient has the disease or not. The images are **unbalanced** which the ones from patient with **pneumonia** being the **73% of the total**. Although the images are black and white, each one of them has **three colour channels**. As we can observe in **figure 1**, the images have **different sizes** that can range **from 500 to more than 2000 pixels**. **Also**, the images have **different** characteristic such as **rotation angle or zoom**.

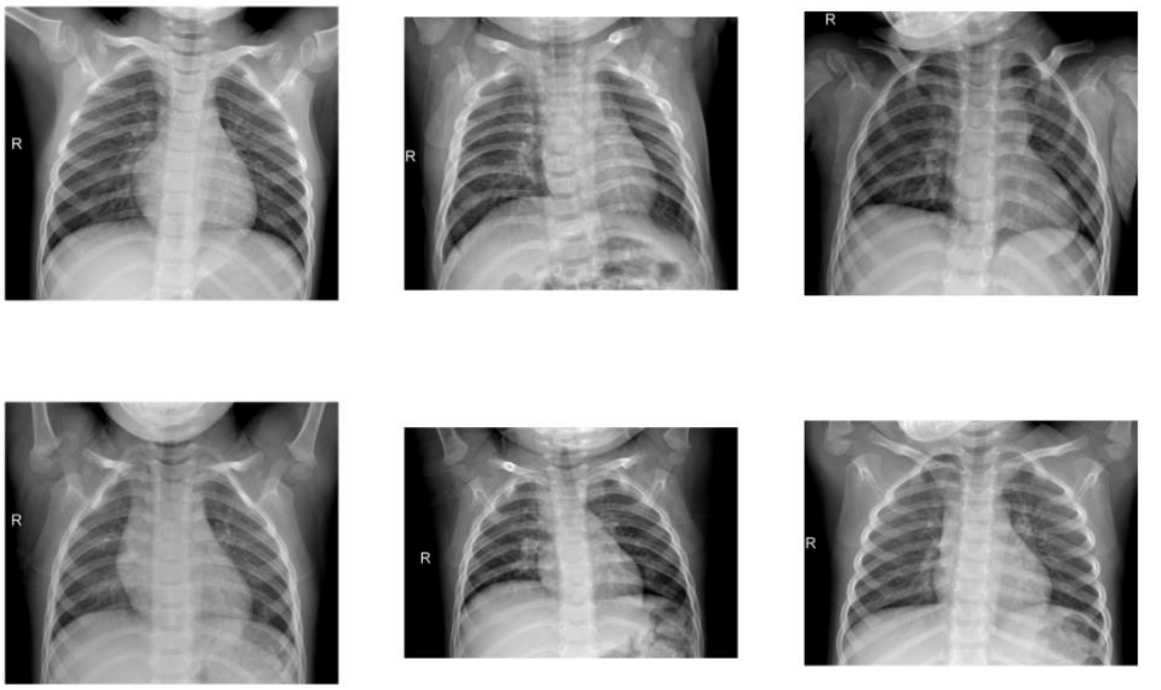

Figure 1. Some of the original chest x-ray images presenting different sizes and characteristics.

Acknowledgements

This dataset is taken from https://data.mendeley.com/datasets/rscbjbr9sj/3.

Reference: https://www.cell.com/cell/fulltext/S0092-8674(18)30154-5

Brief Summary of the Data Pre-processing.

As we saw in the previous section, the images have three channels and the majority of them has very different image size.

The first **transformation** that we will perform on the images will be to **reduce** the images to single **channel** and **set** the **size** of all the images to the **same**. Since we will be using a convolutional neural network, we will select a **square** image (width and heigh with the same

number of pixels). The number of pixels will determine the resolution of the image. The lower the size the smaller amount of information but also the lower computational effort. Selecting a size is a compromise between these two characteristics. In **figure 2** we can see the same image for different sizes.

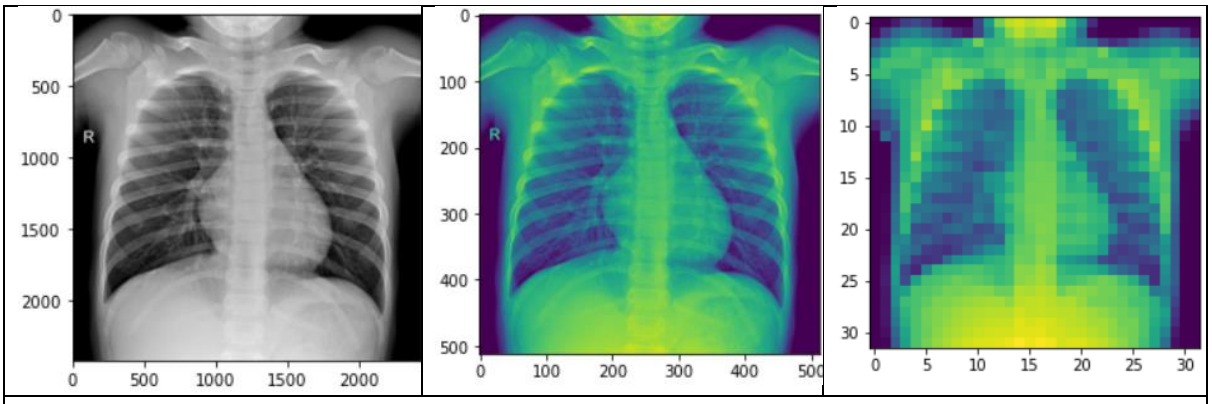

Figure 2. From left to right. A) original (2418x2434x3) B) reduced 1 (256x256x2) C) reduced 2 (32x32x2)

Also, it is a good practice **to normalize the pixel values**. Normally, these values are in a range between 0 and 255. By **dividing all these values by 255** we will have them in a range between 0 and 1 which will be useful to train our model.

Furthermore, we need to transform the **label from categorical to numeric**. Since we have a binary problem, we assign to category pneumonia the value 0 and for the category normal the value 1 by using the LabelEncoder function of scikit-learn.

Finally, we need to **split our data set** into the train and the test set. Also, to train properly the convolutional neural network we will get from the train set a subset to validate the model during the training. So, we will have three different data set with the following distribution by cases (**figure 3**):

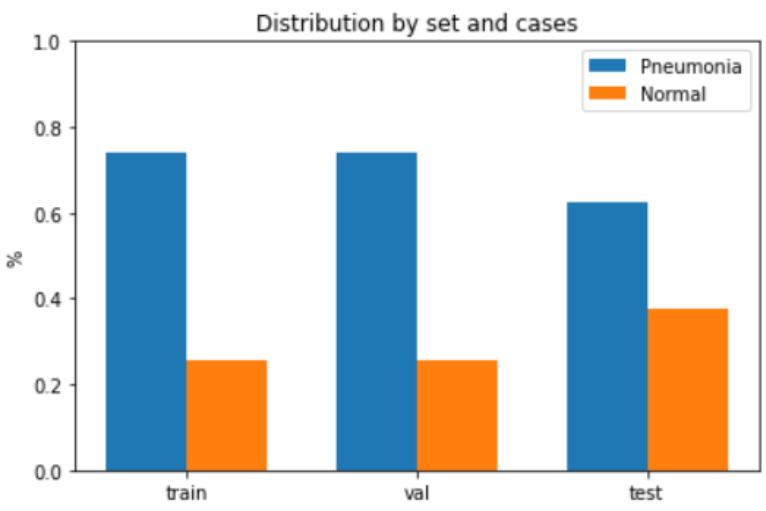

Figure 3. Data distribution by categories for the different sets.

The deep learning models

As we said before, we need to train a **model to classify images into two different cases**. For this purpose, **convolutional neural networks** fit better our needs. We **first** train a **base model** and **then** we will try to **improve** the performance.

The base model

First, we will try with a basic model to set a baseline. In this case we will use an image size of **32x32 pixels**. We will train the model through **10 epochs**, as **optimizer** we will set an **AdaDelta** which does not need to set any parameter. The basic model will consist on **two pairs** of a **convolution layer** with 16 filters and a kernel size of 7x7 followed by a **max pooling layer** with a pool size of 2x2, **then a flatten layer** to prepare the data before a **dense layer** (fully connected), then a **dropout layer before the output layer**. We use a **sigmoid** function as **activation function** for all the layers except for the **output** one that we will use a **softmax** function (see **figure 4**).

```
def base_model():
    base_model = tf.keras.Sequential()
    base_model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=(7, 7), activation='sigmoid', input_shape=input_shape))
    base_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    base_model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=(7, 7), activation='sigmoid'))
    base_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    base_model.add(tf.keras.layers.Flatten())
    base_model.add(tf.keras.layers.Dense(16, activation='sigmoid'))
    base_model.add(tf.keras.layers.Dropout(0.5))
    base_model.add(tf.keras.layers.Dense(2, activation='softmax'))

    return base_model
```

Figure 4. Code for the baseline model.

To **follow the model performance** for the different parameters we will use the **accuracy** and to **test** the model the **confusion matrix** since we are interested on develop a model able to proper classify the images.

For this model we can see in **figure 5** how the loss and the accuracy evolve for the different epochs and on **figure 6** the confusion matrix.

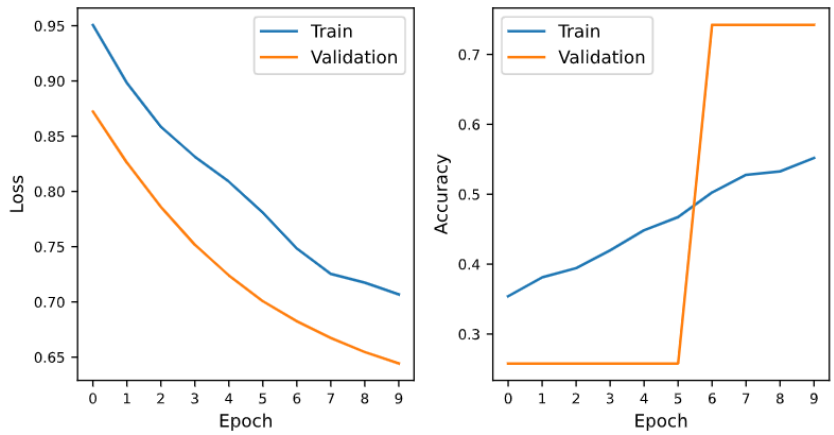

Figure 5. Loss and accuracy vs epochs for train and validation sets

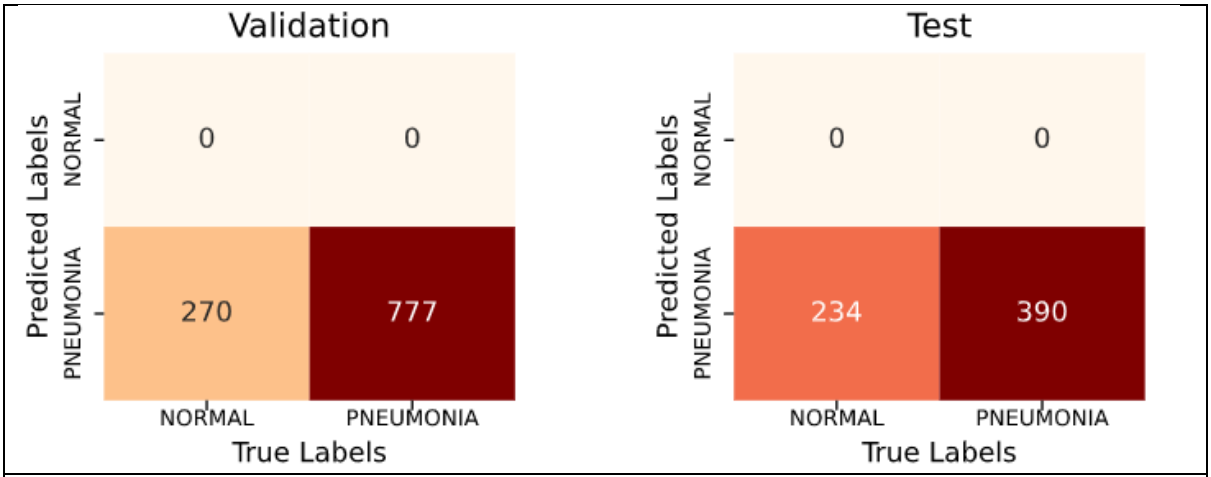

Figure 6. Confusion matrix for baseline model.

Second model

There are many changes that could be done to improve this model. By **increasing** the **size** of the image **and** the **epochs** we can achieve a better result regarding the accuracy in a simple way. Also, we can play with different **optimizers** such as Adaptive moment estimation (**Adam**) which has relatively low memory requirements and works fine with little fine tune of the learning rate parameter. Furthermore, we can use a more suitable **activation function** such as the **ReLU** function. So, for the second model we will perform all this changed by setting **size** to **256 (**larger size make too heavy the computational effort**)**, **epochs** to **30**, the **Adam optimizer** and **ReLU** as activation function. Also, as we saw in figure 6, the unbalance data are creating a problem with the correctly prediction of Normal images having a **large Type-II error**. We can solve this problem by introducing a weight for the different images being the weight for the normal images larger in order to **balance the data set**. Finally, we keep the same layers of the baseline model.

For this model we can see in **figure 7** how the loss and the accuracy evolve for the different epochs and on **figure 8** the confusion matrix.

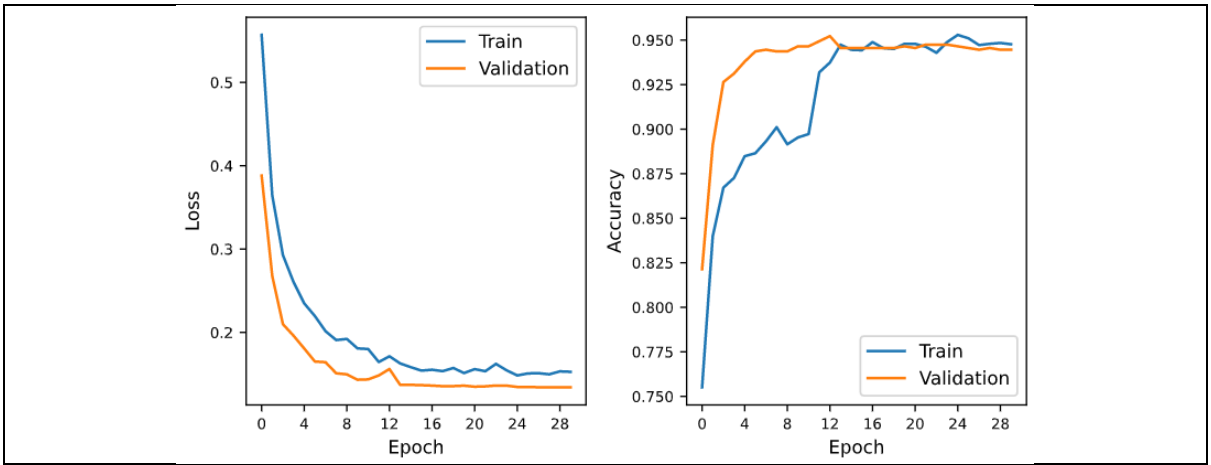

Figure 7. Loss and accuracy vs epochs for train and validation sets

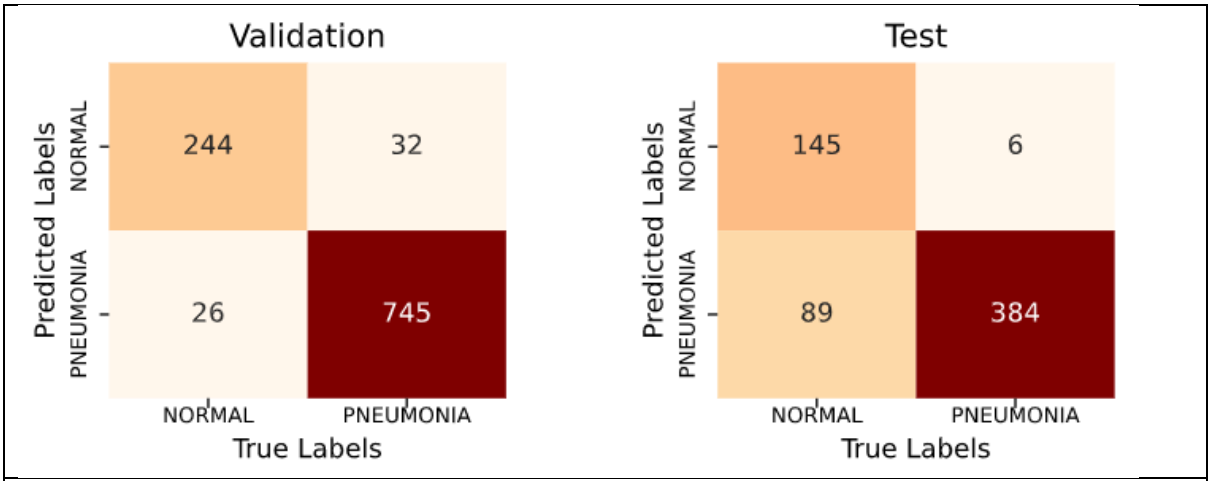

Figure 8. Confusion matrix for the second model.

Third model

The next step to improve the result will be to play with model layers. We can try improve the convolution layers by **using more filters or smaller kernel size**. Also, a **deeper model** will perform better. Thus, we increase the layers by **adding an extra pair of convolution and max pool layers**. In this case, we keep the other parameters used by the second model (size, epochs…)

For this model we can see in **figure 9** how the loss and the accuracy evolve for the different epochs and on **figure 10** the confusion matrix.

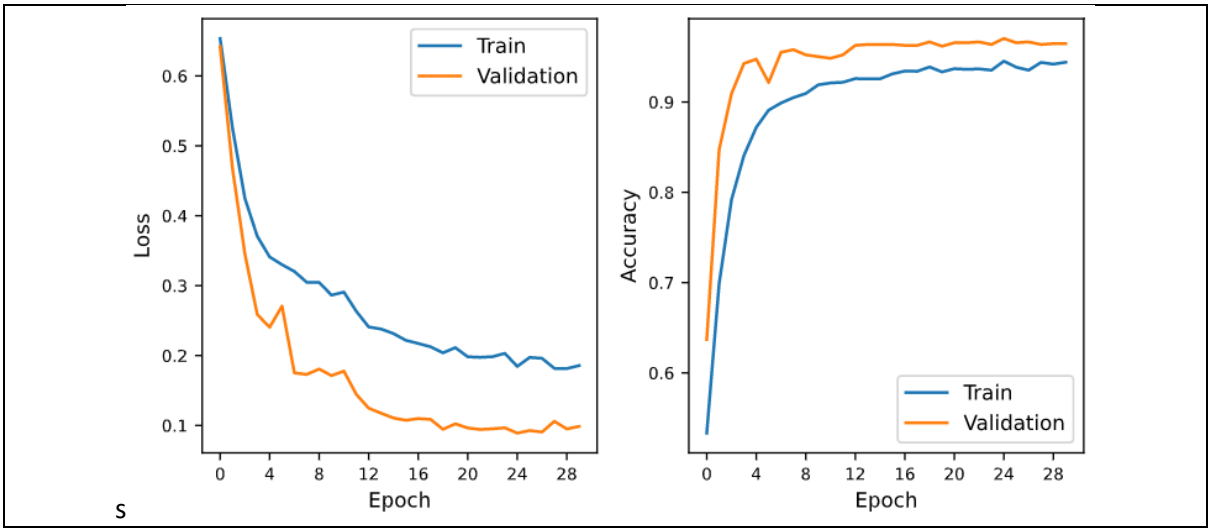

Figure 9. Loss and accuracy vs epochs for train and validation sets

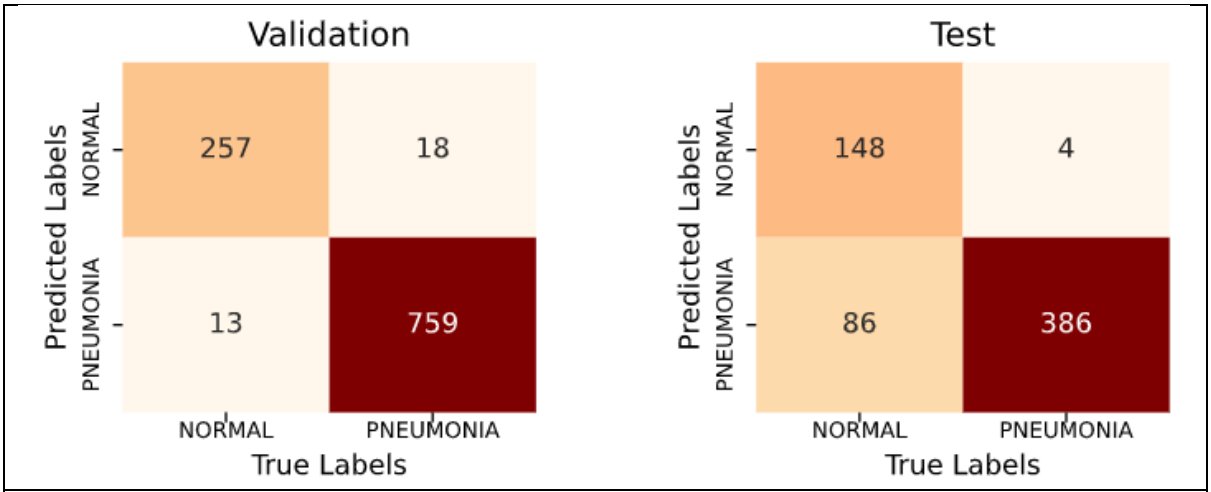

Figure 10. Confusion matrix for the third model.

After having a look on the different model, **we recommend to use a Convolutional Neural Networks** that use an **Adam optimizer** with **three pairs of convolutional/max pooling layers** with a **kernel_size of 3x3** and a **variable number of filters in each convolutional layer**. It seems that an **image size of 256x256 pixels and 15 epochs** is enough to find a good result (**figure 11**).

```
def base_model():
    base_model = tf.keras.Sequential()
    base_model.add(tf.keras.layers.Conv2D(filters=8, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
    base_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    base_model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))
    base_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    base_model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
    base_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    base_model.add(tf.keras.layers.Flatten())
    base_model.add(tf.keras.layers.Dense(16, activation='relu'))
    base_model.add(tf.keras.layers.Dropout(0.5))
    base_model.add(tf.keras.layers.Dense(2, activation='softmax'))

    return base_model
```

Figure 11. Code for the baseline model.

Key findings

As we can see in **figure 10**, we found a **quite good model** that is **able to predict the chest x-ray images with pneumonia in a high percentage** and **also to predict a good percentage of chest x-ray images for patient without the diseases**. We found during this study that a proper number of epochs and a proper image size is important to achieve a good result. The **more epochs we run, the more the model will improve**, up to a certain point. Furthermore, **filters play a huge role** in image recognition. We use filters to transform inputs and extract features that allow our model to recognize certain images. We usually increase the number of filters in a convolution layer the deeper it is in our model. Adding more filters to a convolution layer allows the layer to better extract hidden features. However, this comes at the cost of additional training time and computational complexity, since filters add extra weights to the model Also, we found that **a deeper layer should result in a better model.**

The model can benefit from additional layers. The additional layers allow a CNN to essentially stack multiple filters together for use on the image data. However, similar to building any neural network, we need to be careful of how many additional layers we add. If we add too many layers to a model, we run the risk of having it overfit to the training data and therefore generalizing very poorly.

Next steps

There are **many flaws** that can affect this model. We can try to **improve** the **time performance** as well as the **accuracy** of our model. We can **lower** the **image resolution further without not losing too much information**. Adjust better the **learning rate** of the optimizer, or also try with a different one such as the **stochastic gradient descent** (SGD). We can also play with **different layers combination**, maybe is more interesting to increase the latest dense layer or to try **other architecture such as the CNN VGG16** proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". We could also **increase the number of images** and **use this pretraining model as starting point to train the new model** in the new dataset.