MS Artificial Intelligence

Image Classification of Lung X-Ray Scans

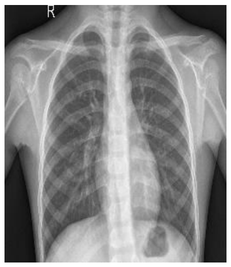
Columbia University in the City of New York

Aneesh Goel (ag4182)

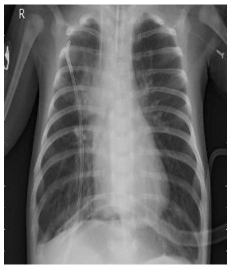
1. Abstract

Given the current scenario of COVID-19, there is a huge scope for artificial intelligence to make a break-through in the healthcare industry. In this paper I explore an application of neural networks in which the aim is to identify whether an individual has pneumonia or not by looking at the x-ray of their lungs. Pneumonia can be caused by either a bacterium or a virus; of which one is the corona virus. Corona - COVID19 virus affects the respiratory system of healthy individual & Chest X-Ray is one of the important imaging methods to identify the corona virus. With the chest X-Ray dataset, I develop a model to classify the X Rays of Healthy vs Pneumonia (Corona) affected patients & this model powers the AI application to test the Corona Virus in Faster Phase. This paper can be further extended to improve the accuracy which could be a head-start into automated testing of diseases (like COVID) by just using x-ray scans.

2. Motivation



Normal



Pneumonia

The two pictures show that it is not possible for a normal human eye to distinguish between the lungs of a normal person and that of a pneumonia infected person. An expert doctor also would prefer to look at the actual x-ray scans and not at images on a computer to give a conclusive answer. The motivation comes from the fact that if the neural network can correctly identify the images then it would become faster to know the disease and take immediate preventive action. If people knew they had corona virus then they would quarantine themselves and the spread of the disease can be controlled in an efficient manner thereby avoiding a pandemic like situation.

3. Data

3.1. Data Collection

The dataset comes from Kaggle [1] which is an online community of data scientists and machine learning practitioners. The data contains 5910 images in JPEG format of which 624 were used to testing and the resting for training the model. The data set was downloaded as a zip file which when unzipped contained the following folders:

- **Train:** This folder had 5309 images each labelled with an image id for example IM-0131-0001.jpeg
- **Test:** This folder had 624 images stored in a similar manner.
- **Chest_xray_Corona_Metadata.csv:** This had the mapping from the image id to the label i.e. from IM-0131-0001.jpeg to Normal
- **Chest_xray_Corona_dataset_Summary.csv:** This excel file had a breakdown of the number of images that belong to each category.

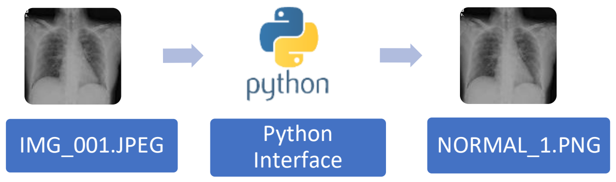
The labels provided were multi-level categories. This meant that if an image belongs to pneumonia category then it can be further classified as stress-smoking, virus and bacteria categories. Further the virus category was broken into COVID-19 and SARS. The complete breakdown is shown in the table below:

Label	virus_category_1	virus_category_2	Count
Normal			1576
Pneumonia	Stress-Smoking	ARDS	2
Pneumonia	Virus		1493
Pneumonia	Virus	COVID-19	58
Pneumonia	Virus	SARS	4
Pneumonia	Bacteria		2772
Pneumonia	Bacteria	Streptococcus	5

3.2. Data Pre-Processing

The first step was to format the images in order to input to the neural network. This was done by creating a dictionary in python which had the keys as the image id and the value as the label. Some of the images were in different formats like jpeg and jpg. This was also made uniform by converting all images to .png format. The unprocessed train images were read from the train folder, processed in python and stored in another folder named labelled_train. Similarly, for the test images. The process is visually depicted below: [2]

[1] Source: https://www.kaggle.com/praveengovi/coronahack-chest-xraydataset
[2] Source: https://towardsdatascience.com/image-classification-python-keras-tutorial-kaggle-challenge-45a6332a58b8



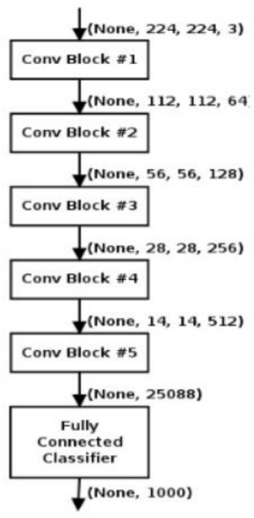
These new labelled images were loaded again in python to train the neural network. To make the images uniform, the height and width was fixed to be 300 pixels. This was seen decided by trial and error to make sure that the information was not lost by making the images too small. The images that were Normal were labelled as 1 and pneumonia as 0. The training images was converted into an array of numbers from 0 to 255 and stored along with their labels in a list to be passed to the neural network.

The train data was split into two sets: training and validation set. 20% of the data was kept for validation and the rest for training. Keras library was used in python to build the network and train the model. It is an open source neural network library which is capable of running on top of TensorFlow and is designed to enable fast experimentation with deep neural networks. Convolutional networks were used as they are currently the standard when it comes to computer vision problems. The model training was done on Google Collab as it provides GPU support and higher computational power (25 GB RAM). The labelled images were uploaded to google drive so that they can be used on Collab.

4. Network

4.1. Architecture

The network architecture that was followed is a popular, effective and simple architecture called the VGGnet [3].



The architecture used in this project was a smaller version of the figure shown, but the interesting thing here is that the filter size increases as we go down the layers. The filters that were used were:

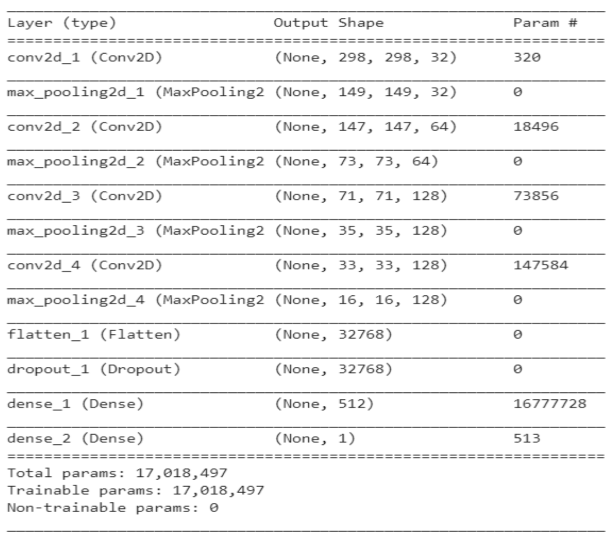
32 → 64 → 128 → 512 → 1 (final)

The first layer is called the input layer and has some important parameters that were set. The hidden layers were all activated by **relu** (Rectified Linear Unit) activation function. The 2D convolution network used a kernel of size [3,3]. This specifies the height and width of the 2D convolution window. The filter size [32] is the size of the output dimension (i.e. the number of output filters in the convolution). The input shape was [300, 300, 1]. The first two dimensions are the size of the image and the last is the number of channels in the image. We have 1 channel here as the images are in grayscale.

We create a sequential model which tells Keras to stack all layers sequentially. We also use MaxPool2D layer whose function is to reduce the spatial size of the incoming features and therefore helps reduce the number of parameters and computation in the network, thereby helping to reduce overfitting. Overfitting happens when the model memorizes the training data. The model will perform excellently at training time but fail at testing time. [4]

1. A **Flatten** layer was also added. Convolution 2D layers extract and learn spatial features which are then passed to a dense layer after it has been flattened
2. Dropout layer with value 0.5 was used to randomly drop some layers and force the network to learn with the reduced architecture. This way, the network learns to be independent and not rely on a single layer thereby reducing overfitting. A value of 0.5 means to randomly drop half the layers.
3. The last layer had an output size of 1 and a different activation function called **sigmoid**. This is because we are trying to detect if an image contains lungs of a normal person or that of a person with pneumonia, i.e. we want the model to output a probability of how sure an image is a normal one and not a pneumonia one. We want a probability score where higher values mean that the classifier believes the image is normal and lower value means it is pneumonia. The sigmoid is perfect for this because it takes in a set of numbers and returns a probability distribution in the range of 0 to 1.

Below is the snapshot of the summary of network. We can see the number of parameters to train (17 million plus) and the general arrangement of the different layers.


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 298, 298, 32)      320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 149, 149, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 147, 147, 64)      18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 73, 73, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 71, 71, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 35, 35, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 33, 33, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 32768)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32768)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               16777728
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 17,018,497
Trainable params: 17,018,497
Non-trainable params: 0
_________________________________________________________________
```

4.2. Compiling

The next step was to compile the model. Three things need to be decided at the compilation step:

[3] Source: https://arxiv.org/pdf/1409.1556.pdf
[4] Source: https://towardsdatascience.com/image-detection-from-scratch-in-keras-f314872006c9

1. We specify a loss function that the optimizer will minimize. Here we are working with a two-class problem so we use **binary cross-entropy loss**
2. The optimizer needs to be chosen as well. The **adam** optimizer was chosen with a learning rate of 0.0001. This was chosen as a part of hyper-parameter tuning. Rmsprop was also tried, however it did not yield good results.
3. The third thing we need to specify is the metric against which we measure our model's performance after training. Since we are doing a classification problem, the **accuracy** metric is a good choice

Finally, before the model was trained the input data was normalized. This insured that the scale of input pixel values has unit standard deviation and a mean of 0.

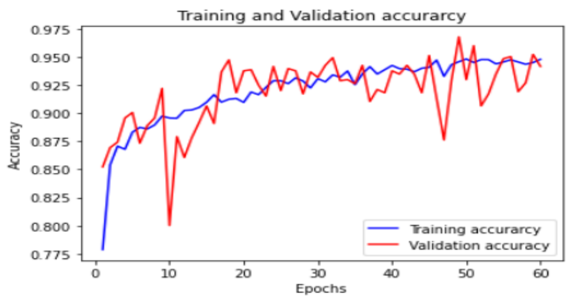
4.3. Data Augmentation

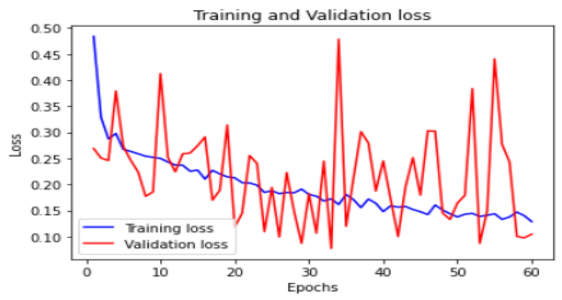
The data size used for training was small so data augmentation was used to create more samples of the images in different forms. To do this, Keras ImageDataGenerator() was used; this quickly sets-up python generators that automatically turn image files into preprocessed tensors that can be fed directly into models during training. It performs the following functions directly:

- Decode the content to RGB grids of pixels (here grayscale)
- Convert these into floating point numbers
- Rescale the pixel values to be between 0 and 1
- It helps to augment the images

Two generators were created one for the training set and one for the validation set. The next step was to train the model. The batch size was kept as 32 and the number of epochs were chosen on the validation set. The number of epochs taken were 60.

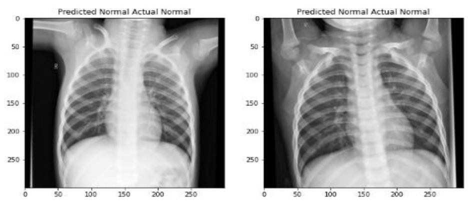
5. Results

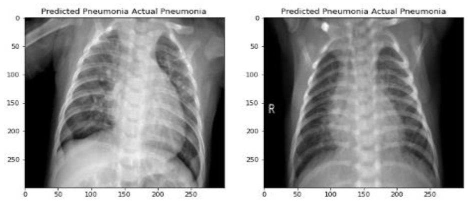




The graphs are the loss and accuracy during training of the model. The accuracy levels out around 50 epochs and we can expect that the weights have been optimized reasonably well. More fine tuning of the number of epochs could be also done in the future.

With this model and its corresponding weights, the test data (also labelled) was loaded and reshaped into the required dimensions (300, 300, 1). Another generator was setup to rescales the test images to be between 0 and 1. As mentioned before, there were 624 test images. The accuracy on the test set came out to be **90.5**. Below is a snapshot of the results where the title of each image represents the actual label as well as the predicted label. (More results in the code)


Predicted Normal Actual Normal
Predicted Normal Actual Normal


Predicted Pneumonia Actual Pneumonia
Predicted Pneumonia Actual Pneumonia

6. Future Scope

1. Multi-level classification can be done provided we have more labelled images for the COVID cases and the other viruses. This could be done with the existing data set as well, however the results would not be good due to scarcity of the data. Having more data, even if it is noise helps the network learn better
2. Fine tuning the architecture by adding more layers, changing the number of filters and kernel size. Playing around with the batch size and number of epochs could also lead to improvement in accuracy

7. References

1. Kaggle data: https://www.kaggle.com/praveengovi/coronahack-chest-xraydataset
2. VGG network: https://arxiv.org/pdf/1409.1556.pdf
3. Blogs: https://towardsdatascience.com/image-detection-from-scratch-in-keras-f314872006c9