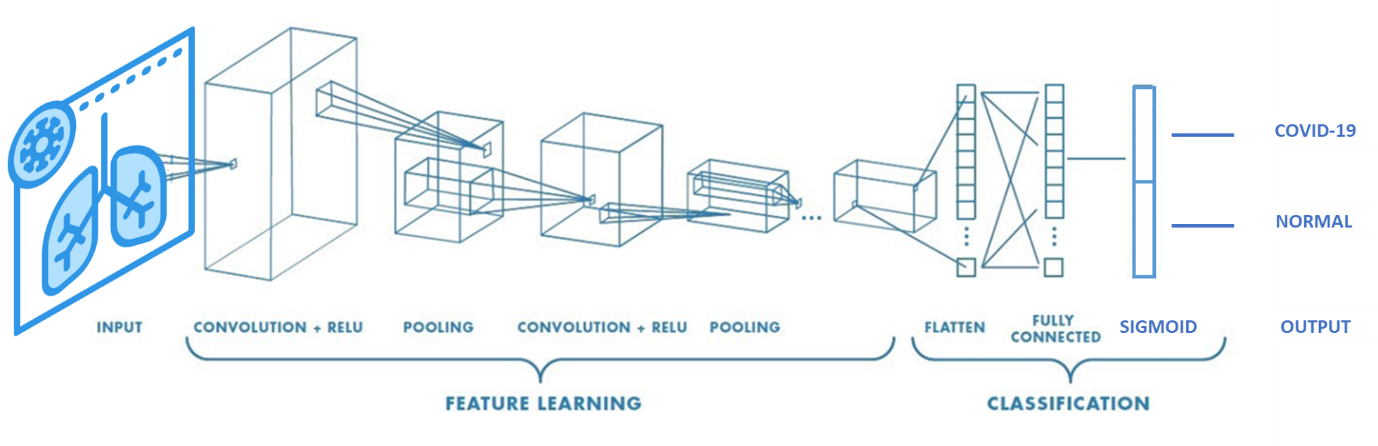
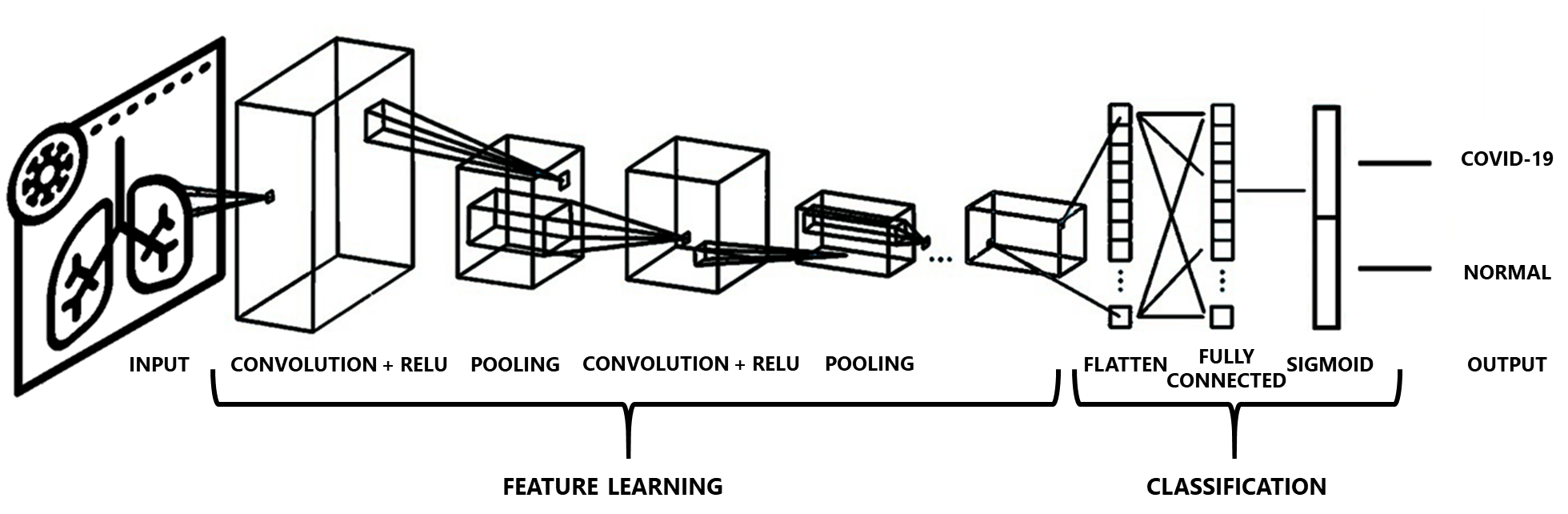
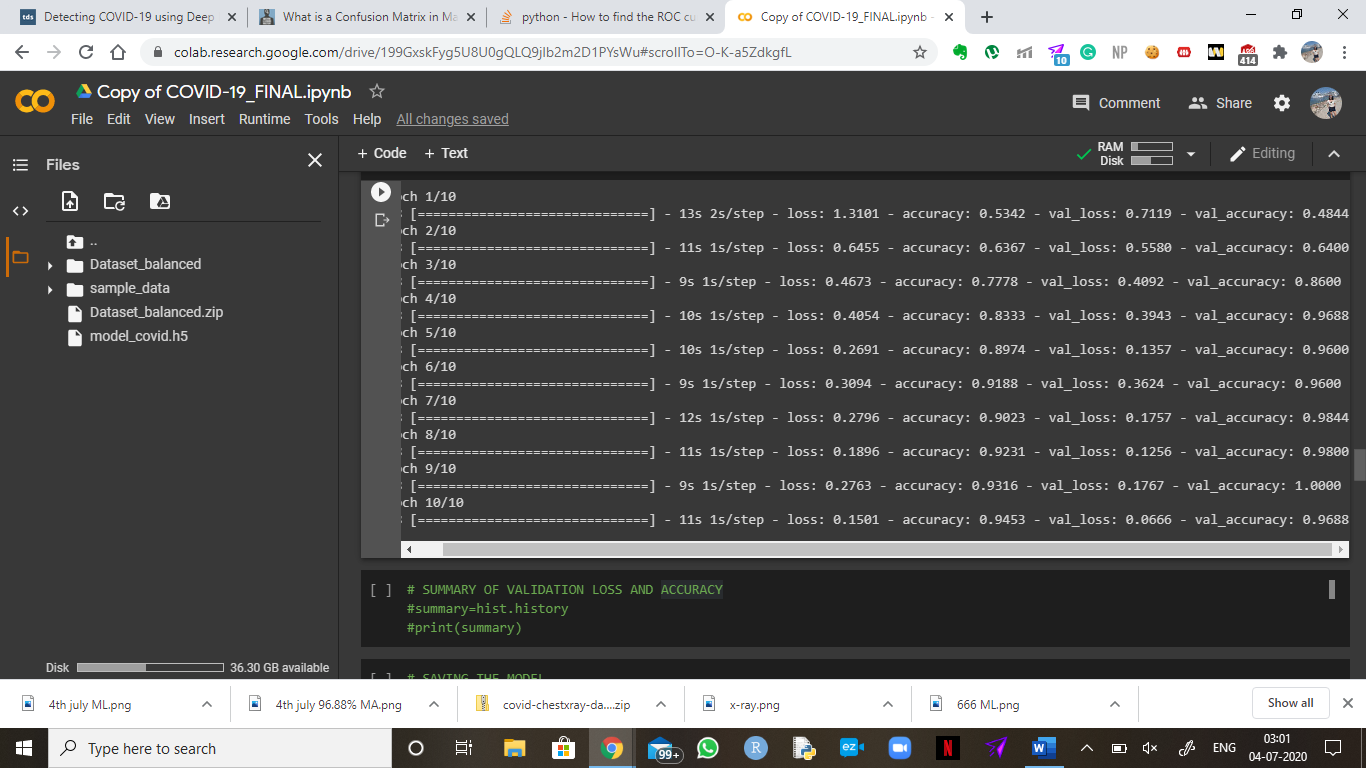
**Ray data Pointers of the Dissertation**

1. The situation of COVID-19 has affected all the countries worldwide and there is an urgent need of is doing rapid testing and evaluation of the results.
2. We use .jpg files of chest x-rays of patients having COVID-19 and normal patients
3. Advantage of using chest x-rays are that a) testing is costly (around Rs. 5000) and it takes a long time for detection of the results. b) We can see the spread of the disease in the lungs of the patients.
4. Coronavirus affects the bronchioles and lung airwaves from which the air passes. It blocks the air sacks and can analyse the spread of the disease with x-rays. This shows whether the patient is affected mildly or severely.
5. We can use Image Classification models on the x-ray images like CNN and also use segmentation techniques to find which particular regions have been infected.
6. We use COVID-19 images from GitHub, in which the data has been collected from a team of doctors regularly. We only consider the COVID-19 positive cases.
7. To build a classifier we need positive and negative images as well. We take a very popular dataset from Kaggle for Normal x-ray images.
8. We combine the dataset from Kaggle and GitHub.
9. For GitHub, we have a metadata file which gives patient information like age, sex, diagnosis, etc We only consider COVID-19 images.
10. One cannot easily detect which image is positive or negative for COVID-19 using the naked eye. Hence, we deploy machine learning techniques to detect patterns, blurriness, textures present and see minute details in the image.
11. Since this image has fine grained features, it is more difficult than normal problems. This is a fine-grained visual recognition problem.
12. We build a classification model using Convolutional Neural Network and check its validation accuracy.
13. After building the model, we can see where all has the infection spread in the lungs using Grad-Cam and Saliency Maps.
14. Even though, the model does give a good accuracy score, the model gives a xx% accuracy, it may have cons to deploy the model since there could be some major consequences like leaving xx positive cases out of 100 not quarantined leading to the spread of the disease or vice versa.
15. When it comes to medical imaging, one must be super cautious and have a good accuracy score since this is a question of human life.
16. Architecture: in the convolutional operation, a filter will slide through the entire image and it will highlight the regions where the features are present.
17. Once we obtain the image, we can extract the features and once we know the features, we can do the binary classification; whether the patient has COVID-19 or not.
18. CNN model has 2 parts:
19. Feature extraction: This is built on a convolutional and pooling layer
20. Dense layers: This is for classification
21. The best thing about deep learning is that we do not know how to classify the images, the model will learn from the provided data and will learn to classify.
22. The script for data creation is written in Jupyter Notebook. We use pandas, os and shutil to create the dataset.
23. The images along with the metadata file is uploaded into the jupyter notebook where we apply 2 filters, i.e. finding==” COVID-19” and view==”PA” which is the post anterior view, i.e. the front view of the chest x-ray.
24. We consider all of the images from the Normal chest x-ray dataset.
25. The dataset is divided into training and validation with the ratio 80:20.
26. Since the image data is huge, we create a zip file and upload the images in google colab.
27. The advantage of google colab is that it provides all the features of Jupyter Notebook and additional benefits like it is easily connected to Google Drive and uses runtime as GPU so that there is no load on the CPU of the system.
28. We start building the model and define a path for the training and validation image folder.
29. We import libraries like numpy, matplot.lib, keras, etc which uses TensorFlow in the background.
30. We build a CNN model using Keras. It is a sequential model having multiple layers. It is a layered architecture having filters.
31. The 1st layer is a 2D convolutional layer with 32 filters. The initial layers are small in the beginning because the lower layers detect features in small parts of the images and can find small patterns in the image. As we go deeper into the layers, the receptive field of the CNN layer increases. The kernel size is 3 x 3 which is a standard choice. Activation of Relu layer is used. Since this is the first layer, we specify the input size which is (224, 224, 3).
32. 2nd Conv layer with 64 filters having kernel size = (3,3) and activation function of relu is used.
33. Adding a pooling layer with default pooling size (2,2). Max pooling increases the receptive field of the layer.
34. We do not use VGG because it has 140 mil parameters leading to overfitting because our dataset is small.
35. X-ray images in the 1st conv layer is reshaped into (224,224) with 3 channels as it is an RGB image.
36. If we closely notice the x-ray images, the images aren’t black and white but RGB in nature, some pictures have a blue or yellow tone.
37. We use 2 layers of 3 x 3 rather than 1 layer of 5 x 5 kernel size because a) it increases non linearity and b) there are less parameters and hence avoiding overfitting.
38. As we go deeper into the model layers, the model will be able to detect higher level of features in images.
39. We add dropout to check for overfitting.
40. 3rd conv layer added with 64 filters with same pooling layer and dropout.
41. 4th conv layer added with 128 filters with same pooling layer and dropout.
42. Adding a flatten to convert 2D conv layers into 1D and then adding a fully connected layer.
43. Output layer has 1 filter and hence we use the sigmoid function.
44. We compile the model with binary entropy loss and Adam optimiser with accuracy metrics.
45. Model summary gives us 56 lakh parameters in total which we train from the scratch.
46. Input of 224 x 224 becomes 222 x 222 after the 1st convo layer and before the flatten layer, we have the output shape as 26 x 26 or 26 x 128.
47. As we go deeper into the model, we increase the number of channels (receptive field). More distinct patterns can be found as we dive deeper into the layers.
48. The inbuilt Keras Image Generator library is used to train the dataset. The data is rescaled for normalisation (1/255). This gives an easier convergence.
49. Sheer and zoom augmentation is added which allows to take random crops from images and those will be like zooming into the images which is around 20% magnitude of the image. Horizontal flipping is allowed but not vertical since the x-ray should not be inverted for accurate results. On the test data, the image is rescaled with the in-built library Image DataGenerator.
50. We use flow from directory with target size (224,224) to reshape the image and batch size = 32 using binary class (coved and normal). The same is done for validation data.
51. 224 x 224 is a standard choice and most ImageNet problems are solved using this input size. If we keep a bigger size, it will be difficult to train the model and if we keep a smaller size, it will be difficult to capture fine grained features.
52. This model is simpler than ImageNet as it uses less parameters.
53. ImageNet will not give good results as we are dealing with Chest x-rays and would have to carefully tune the parameters which is tedious.
54. Shortcomings of the dataset:
55. It is a different kind of dataset since it is very medical in nature.
56. Small dataset.
57. We cannot use transfer learning since our dataset must have at least 2500-3000 images and the parameters must be fine-tuned to get good results.
58. Future work: Implement Grad-Cam and Saliency Maps to find the regions where COVID-19 affects the chest.
59. Predict and Use confusion matrix to get accuracy.
60. This dissertation is only for educational and research purposes, not for medical purposes.







Epoch 1/10

8/8 [==============================] - 13s 2s/step - loss: 1.3101 - accuracy: 0.5342 - val\_loss: 0.7119 - val\_accuracy: 0.4844

Epoch 2/10

8/8 [==============================] - 11s 1s/step - loss: 0.6455 - accuracy: 0.6367 - val\_loss: 0.5580 - val\_accuracy: 0.6400

Epoch 3/10

8/8 [==============================] - 9s 1s/step - loss: 0.4673 - accuracy: 0.7778 - val\_loss: 0.4092 - val\_accuracy: 0.8600

Epoch 4/10

8/8 [==============================] - 10s 1s/step - loss: 0.4054 - accuracy: 0.8333 - val\_loss: 0.3943 - val\_accuracy: 0.9688

Epoch 5/10

8/8 [==============================] - 10s 1s/step - loss: 0.2691 - accuracy: 0.8974 - val\_loss: 0.1357 - val\_accuracy: 0.9600

Epoch 6/10

8/8 [==============================] - 9s 1s/step - loss: 0.3094 - accuracy: 0.9188 - val\_loss: 0.3624 - val\_accuracy: 0.9600

Epoch 7/10

8/8 [==============================] - 12s 1s/step - loss: 0.2796 - accuracy: 0.9023 - val\_loss: 0.1757 - val\_accuracy: 0.9844

Epoch 8/10

8/8 [==============================] - 11s 1s/step - loss: 0.1896 - accuracy: 0.9231 - val\_loss: 0.1256 - val\_accuracy: 0.9800

Epoch 9/10

8/8 [==============================] - 9s 1s/step - loss: 0.2763 - accuracy: 0.9316 - val\_loss: 0.1767 - val\_accuracy: 1.0000

Epoch 10/10

8/8 [==============================] - 11s 1s/step - loss: 0.1501 - accuracy: 0.9453 - val\_loss: 0.0666 - val\_accuracy: 0.9688