CMPE 255 Project Report – Team 2

Project Title: Detecting Tuberculosis from Chest X-ray images

GitHub Link: https://github.com/himayu-m/CMPE-255-Team-2

Team Members:

Himayu Mahnot, 015242357

Ruchita Dinesh Entoliya, 015331966

Shivani Sanjay Degloorkar, 015237664

Vishalsinh Hareshwarsinh Gadhvi, 015231450

Section 1 Introduction

Motivation

Tuberculosis (TB) is caused by bacteria called Mycobacterium tuberculosis, which mainly attacks the lungs. The bacteria spreads when an infected person coughs or sneezes. Majority of the people have TB undetected for prolonged periods, which results in the risk of transmitting Tuberculosis to other people. Thus, it is very essential for early diagnosis and treatment of Tuberculosis.

Currently, TB is one of the top 10 causes of death and the leading cause from a single infectious agent, it is crucial that we accurately identify TB at the early stages. Chest radiography is an essential tool for the early detection of TB, and therefore fundamental. The X-ray images generated are used to detect infections in the posterior-anterior region of the chest. The infection may appear anywhere in the lungs and may differ in shape, size, cavitation and density. Using deep learning algorithms, it is possible to accurately detect TB and quickly classify a wide range of medical X-ray images without human supervision.

Also, this work can be expanded to produce useful and impactful study on COVID-19, which may facilitate braving this pandemic.

Objective

Given a chest X-ray dataset, our goal is to train a classification model to learn the different features of a normal and a tuberculosis chest X-ray. Our final objective is to create an application that will use the trained models to predict whether a given chest X-ray has tuberculosis or not.

Section 2 System Design & Implementation details

Algorithms Selected

Deep learning with Convolutional Neural Network (CNN) architectures work well in medical image classification tasks. CNN captures the spatial features from an image. These features correspond to the arrangement of pixels and help detect the relationship between them in an image. Thus by using a CNN, we can identify the object accurately, the location of an object, and also its relation with other objects in an image. The main advantage of CNN compared to other neural networks is that it automatically detects the important features without any human supervision.

For training, the deep learning algorithm uses the training set to identify the image properties that, when used, will result in the correct classification of the image—that is, depicting if the patient has Tuberculosis or not. We used four different CNN architectures to perform transfer learning and train using our dataset:

- DenseNet201
- InceptionV3
- VGG19
- InceptionResNetV2

For our final application, we are predicting using each model and then taking a weighted vote to give us an even better accuracy. The weights sum up to 1, and were manually assigned based on the training and validation accuracy of each architecture. The weights are defined as follows: DenseNet201 (0.3), InceptionV3 (0.3), VGG19 (0.2), InceptionResNetV2 (0.2). If the weighted sum is greater than 0.59 (sum of the weights of the two highest accuracy models), then the final prediction is Tuberculosis otherwise Normal.

Technologies and Tools used

IPython Notebooks on Google Colab with GPU accelerator was used to train the dataset. The following libraries were used:

- tensorflow and keras: used to import different CNN architectures, and fit the imported models to our training dataset.
- ImageDataGenerator from keras.preprocessing.image: used to read and augment our dataset which was stored on Google Drive
- sklearn.metrics and mlextend.plotting: used to plot the confusion matrix
- matplotlib: used to plot the accuracy and validation accuracy while training

Our final application was a Python script that used the following libraries:

- sys: to read in arguments (image path) from the command line
- PIL.Image, requests, io: used to load the images from a given path or a web url
- numpy: used to store the image pixels in an array
- tensorflow: used to load the pretrained models and make a prediction

Architecture-related decisions

We added extra layers to each of the pretrained model to customize it to fit our requirements:

- GlobalAveragePooling2D: to downsample the number of features that are being detected. This reduces the chances of overfitting the data
- BatchNormalization: used to normalize the outputs of a layer which can be used to increase the speed at which the model learns.
- Dropout: regularization technique that is used to prevent overfitting by randomly 'switching off' neurons
- Dense: a fully connected layer with varying number of units with the activation as 'ReLu'
- Output layer: with 1 unit and activation as sigmoid since it is a binary classification

System Design and Data Flow

These are the steps performed inside the python notebook:

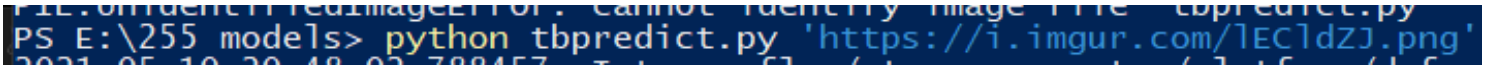
- Load the images into train/val from drive using ImageDataGenerator.flow_from_directory
- Import the CNN architecture and set the layers to untrainable.
- Add extra layers to the pretrained model
- Compile the model and call model.fit on the training dataset
- Plot accuracy graphs and confusion matrix
- Save the model as a '.h5' file

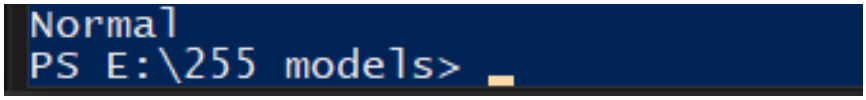
Our final application is a python script that takes in the path to the image or a web url to classify the chest X-ray. These are the steps performed inside the application:

- Load all 4 saved '.h5' files using tensorflow.keras.models.load_model()
- Load the image into a numpy array and preprocess the image to size 224*224 since that was the input shape to the pretrained model
- Predict using each model
- Final prediction is a weighted sum of each model

Use Cases

Example command is: $python tbpredict.py <path to file>
<path to file> accepts both image URL and system path





Section 3 Experiments and Proof of Concept Evaluation

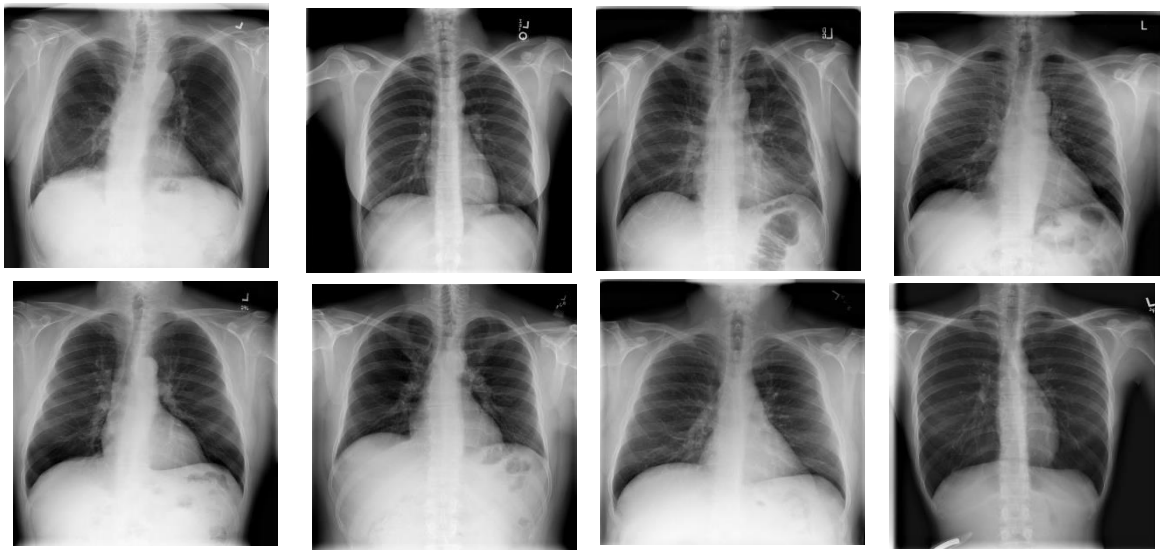
Datasets Used

The dataset was sourced from: https://ieee-dataport.org/documents/tuberculosis-tb-chest-x-ray-database#files

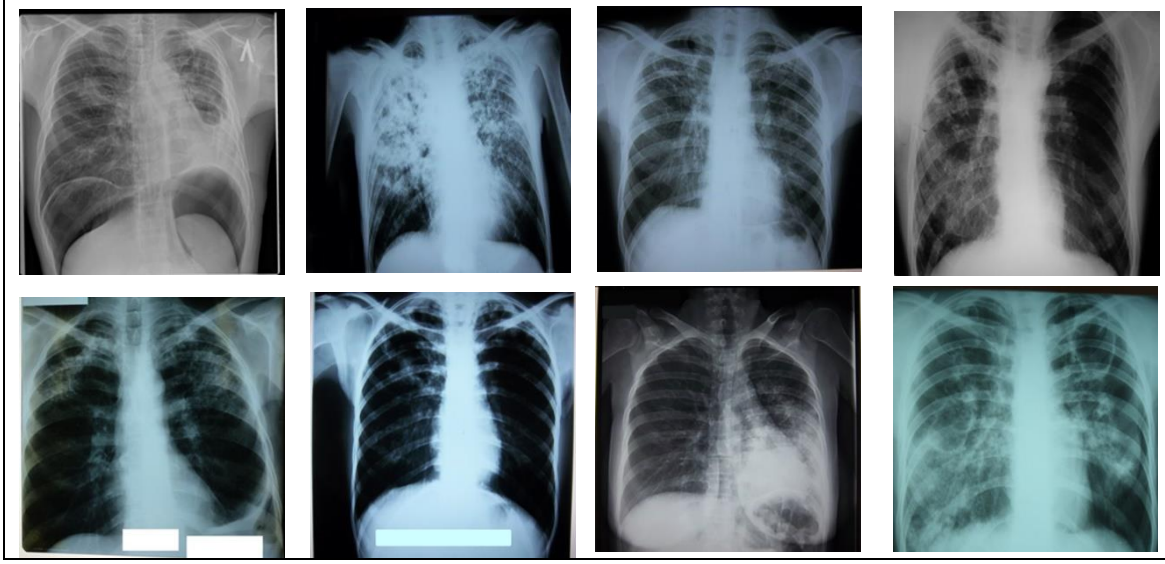
The dataset chosen contains 7000 labelled images, with 3500 belonging to the normal class, and 3500 belonging to the Tuberculosis positive class. The input size is 512*512, and with 7000 images, we will have approximately 1835008000 data points (pixel values) to learn and train from.

Some sample pictures from our dataset:

Normal Images



Tuberculosis Images



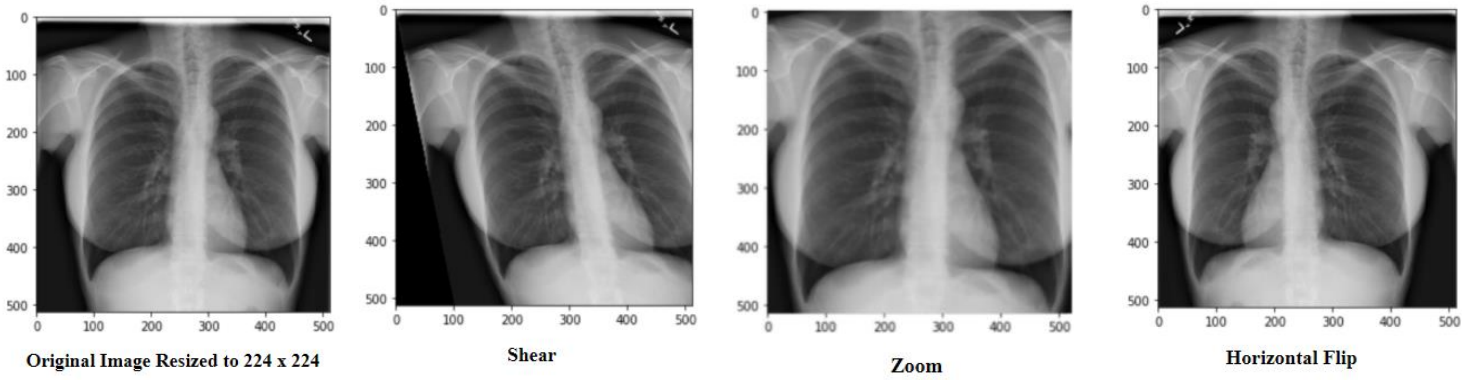
Preprocessing

Using the ImageDataGenerator class from Keras, we were able to load images from google drive and randomly augment images. Since we only used a small amount of the data, the random transformations allow our model to generalize much better. The images were in grayscale but in order to use most of the pre trained models, we had to pass rgb channels so had to transform images into 3 channels, i.e., color_mode = 'rgb'.

The following transformations were applied to our dataset:

- Target Size: Different images had different sizes, so we used this parameter to resize each image to (224, 224). This allowed us to pass the same parameter to each model.
- Rescale: This parameter normalizes the image, so each pixel value is between 0 and 1 instead of 0 and 255.
- Shear Range: This performs a shear operation on the image, in the counterclockwise direction.
- Zoom Range: This parameter randomly zooms the image in or out.
- Horizontal Flip: Randomly flip images along the y-axis.

Results from preprocessing the image:



Original Image Resized to 224 x 224 | Shear | Zoom | Horizontal Flip

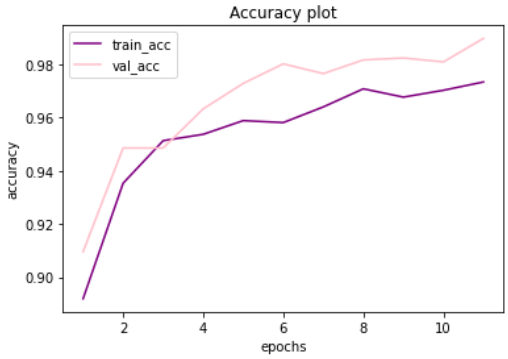
Methodology Followed

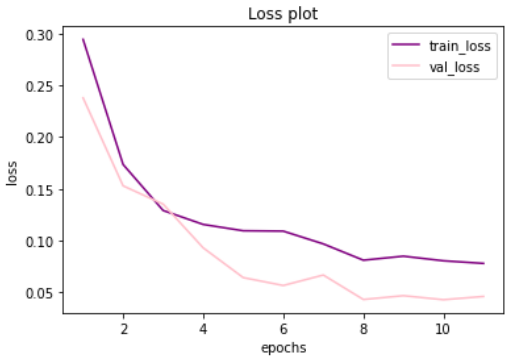
There was no explicit test dataset provided, so we took out 100 images from each class (total 200 images) as our test dataset for our application. The remaining 6800 images were split using an 80/20 train/val split to train and validate our model. 5440 images were used for training and 1360 images were used for validation.

Evaluation Methods

Following are the graphs for training and validation accuracy as well as the confusion matrix (evaluated on validation data) for each CNN architecture:

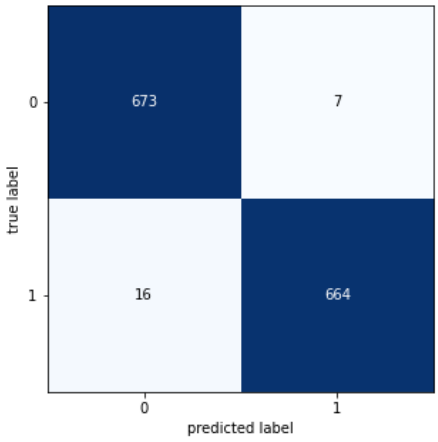
DenseNet201



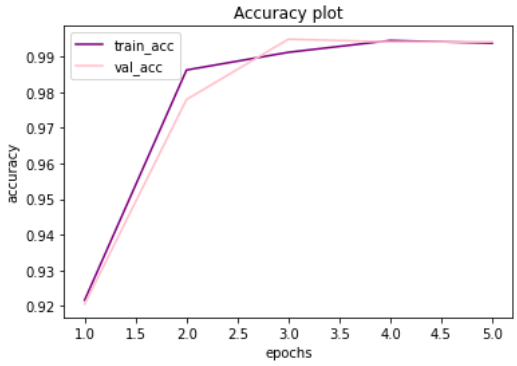


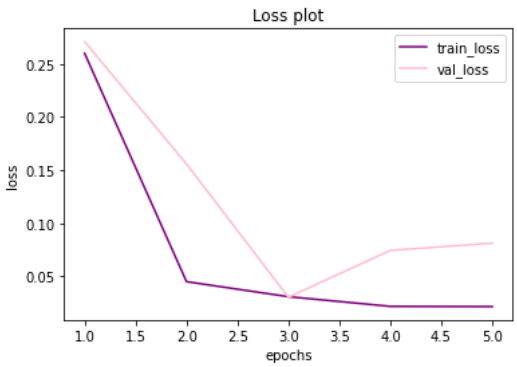
0.9830882352941176

	precision	recall	f1-score	support
0	0.98	0.99	0.98	680
1	0.99	0.98	0.98	680
accuracy			0.98	1360
macro avg	0.98	0.98	0.98	1360
weighted avg	0.98	0.98	0.98	1360



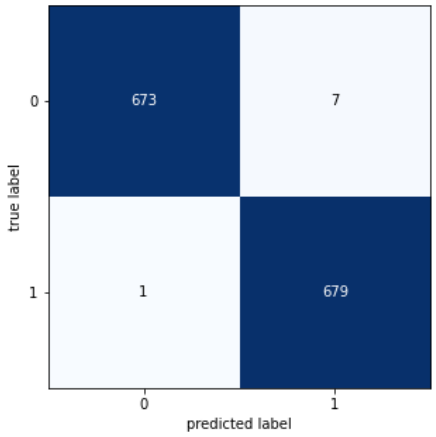
InceptionV3



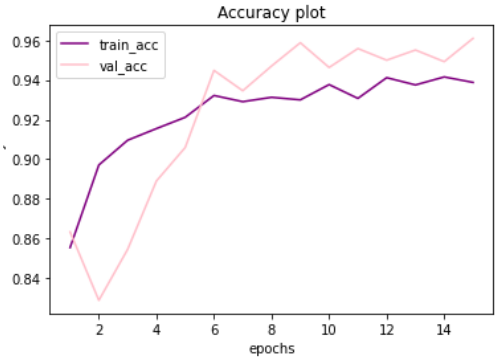


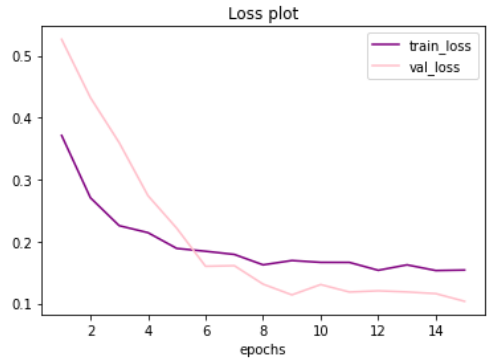
0.9941176470588236

	precision	recall	f1-score	support
0	1.00	0.99	0.99	680
1	0.99	1.00	0.99	680
accuracy			0.99	1360
macro avg	0.99	0.99	0.99	1360
weighted avg	0.99	0.99	0.99	1360



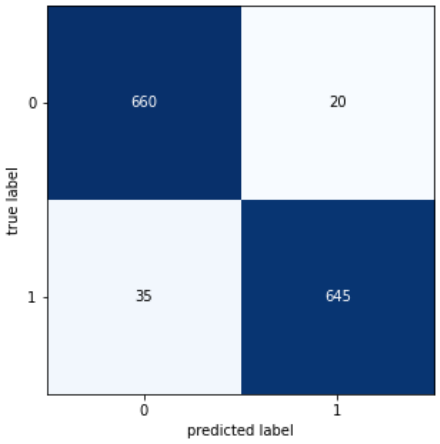
VGG19



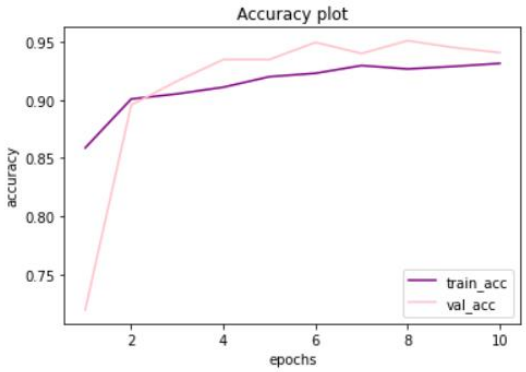


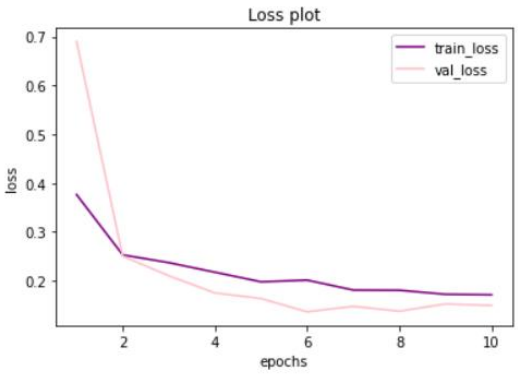
0.9595588235294118

	precision	recall	f1-score	support
0	0.95	0.97	0.96	680
1	0.97	0.95	0.96	680
accuracy			0.96	1360
macro avg	0.96	0.96	0.96	1360
weighted avg	0.96	0.96	0.96	1360



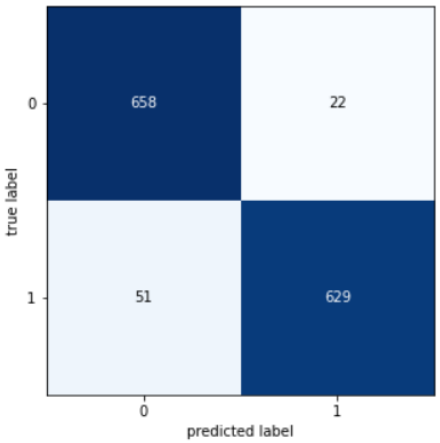
InceptionResNetV2





0.9463235294117647

	precision	recall	f1-score	support
0	0.93	0.97	0.95	680
1	0.97	0.93	0.95	680
accuracy			0.95	1360
macro avg	0.95	0.95	0.95	1360
weighted avg	0.95	0.95	0.95	1360



Analysis of Results

All four models, performed really well on the validation set with the accuracy being at least 95%. All four models, also had high values for precision, which means that the number of false positives were very low. Similarly, they also had a high recall value, which suggests that the model didn't classify too many examples as false negatives.

However, there was a difference in performance of each model. InceptionV3 was the best performing model, with only 1 image classified as false negative and 7 images classified as false positive out of a total of 1360 images. DenseNet201 was the second-best performing model, with 16 images classified as false negatives and 7 images classified as false positive. Using VGG19, we had 35 images classified as false negatives and 20 images classified as false positives. InceptionV3 was the worst performer with 51 images classified as false negative and 22 images classified as false positive. We used this evaluation result to assign manual weights to each model, which will be used in calculating the combined vote for our final application.

Section 4 Discussion and Conclusions

Decisions Made

Our initial plan was to create a csv file out of all images in the dataset and then use it to read in images. After reading the images we converted it to grayscale and resized to 224x224 as a part of dimensionality reduction. However, all the pretrained models required an input which should exactly have 3 color channels. We also planned to use a Linear Regression model to automatically give us the weights for the voting system but we had to discard it since the implementation was complicated.

We switched our approach from using csv file to read images directly from the directory using flow_from_directory(). We also uploaded our dataset to Google Drive instead of using it from the local directory. We also decided to manually assign weights instead of using a Linear Regression model.

Difficulties Faced

Image processing is well suited for GPUs. When doing image processing, we need fast access to pixel values. Therefore, the hardware for accessing and manipulating pixels needs to be well optimized. All of our team member's systems lacked a good GPU for processing 7000 images. To overcome this problem, we switched to Google Colab environment with GPU which made our image processing real fast.

Another difficulty we faced was that not all images had 3 color channels, which is why our initial approach was causing issues. Our problem was solved by switching to flow_from_directory()

Things that worked

- Training on Google Colab using a GPU accelerator
- Loading images from Google Drive using flow_from_directory()
- Majority vote system to give us a better accuracy

Things that didn't work well

- Training on local computer using CPU
- Using a csv file for train and test data

Conclusion

Our goal was to train and predict chest X-ray images with tuberculosis. For this we used four different CNN architectures. We trained our models on Google Colab with a GPU accelerator and saved them as '.h5' file. By using a python script, we took a weighted vote of prediction from each model, and by doing this we were able to do a binary classification on the chest X-ray with very high accuracy.

Section 5 Project Plan/Task Distribution

The following tasks were assigned and completed by each team member:

- Himayu
  - Wrote the code for plotting confusion matrix
  - Trained and tuned DenseNet201 model
  - Created the script for the final application
  - Worked on section 2 of the report
- Ruchita
  - Created train.csv and test.csv files
  - Created a function to read the image from the csv files and convert it to grayscale and reduce the image size
  - Wrote the code for using ImageDataGenerator
  - Trained and tuned InceptionV3 model
  - Worked on section 1 of the report
- Shivani:
  - Wrote the code to fit the model, and plot the accuracy and validation graphs
  - Trained and tuned VGG19 model
  - Worked on section 4 of the report
- Vishal:
  - Wrote the code to import and build a pretrained CNN architecture
  - Trained and tuned InceptioResNetV2 model
  - Worked on section 3 of the report

References:

1. https://ieee-dataport.org/documents/tuberculosis-tb-chest-x-ray-database#files
2. https://ieeexplore.ieee.org/document/9224622