CSE 676: Deep Learning, Summer 2024

Instructor: Alina Vereshchaka

Final Project Proposal

Title: Enhancing Pneumonia Diagnosis based on Chest X-rays with Deep Learning

Short Summary:
Pneumonia is a life altering respiratory disease, which if not treated properly can lead to death. This project aims to enhance the accuracy and reliability of pneumonia diagnosis using chest x-rays using deep learning techniques. We aim to build a robust model with high precision and recall in identifying pneumonia using Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs).

Objectives:

- Develop a deep leaning model to classify chest X-ray images into +ve and -ve categories accurately.
- The model's performance can be improved through GAN-based data augmentation to address the issue with limited labeled data.
- Evaluate the model's performance against existing benchmarks and ensure it meets clinical standards for diagnostic accuracy.

Methodology:

- Build a Convolutional Neural Network to detect pneumonia based on chest X-rays.
- Tune the hyperparameters for the said model to optimize the performance.
- Applying Deep Ensemble methods like VGG16, ResNet50 etc. with different architectures and combine the predictions of these models for final prediction.
- Generation of synthetic X ray images using GANs will be considered if the data is imbalanced thereby enhancing the robustness of the model.
- Fine tune the models and based on the results of each model, best model will be picked for this project.

Evaluation:

Qualitatively,

- Confusion matrix – a clear diagonal dominance indicating correct classification
- ROC Curve: Area under the curve will be a significant indicator of the model performance

Quantitatively,

Parameters like high Accuracy, Precision, Recall, F1 Score are indicative of good model performance.

Dataset:

This dataset is available in kaggle.

References:

1. GAN-based Data Augmentation for Chest X-ray Classification – Shobhita Sundaram, Neha Hulkund
2. Diagnostic accuracy of deep learning in medical imaging: a systematic review and meta – analysis – Aggarwal et.al.
3. Interpreting chest X-rays via CNNs that hierarchical disease dependencies and uncertainty labels – Hieu H.Pham et.al.
4. https://developers.google.com/machine-learning/gan/gan_structure - overview of generative adversarial network (GAN)
5. Pneumonia detection in chest X-ray images using an ensemble of deep learning methods – Kundu et.al.
6. A Review of Deep Learning Based Medical Image Segmentation methods – Xiangbin Liu et.al.

CSE 676: Deep Learning, Summer 2024

Instructor: Alina Vereshchaka

Checkpoint - 1 Report

Team Members
Hariharan Sriram (hsriram)
Swanith Ambadas (swanitha)
Manish Bikumalla (manishbi)

Introduction:

The aim of this project is to leverage deep learning models like CNN and apply them to chest X-rays image data to classify whether a person suffers from pneumonia or not. This method promotes an improved detection/diagnosis and further aids physicians in treating the patients thereby enhancing the health outcomes.

Dataset:

The Chest X-Ray Images (Pneumonia) dataset contains **5,856** chest radiograph images in PNG format, categorized into 2 classes: Normal and Pneumonia. It aims to help us in the development of diagnostic models for pneumonia. The dataset is split into training and testing and validation subsets. Each image is labeled to identify the presence and type of pneumonia, if present. The dataset is widely used for training machine learning algorithms to improve the accuracy of pneumonia detection from chest X-rays, fostering advancements in medical imaging analysis. The dataset consists of a total of 5856 images of *Normal*(1583*) and Pneumonia(4273*).

Viewing the dataset:

The dataset has been uploaded in the UBBox, both the original, concatenated and splitted.

Data Augmentation:

Data augmentation has been performed on this dataset by introducing some transformations like rotation, resizing, flipping and cropping. This further helps the model to recognise various patterns and features under various conditions. Further helps the model to perform well on new and unseen images. This has been done using the below code:

```
transform = transforms.Compose([
    transforms.Grayscale(),  # Convert images to grayscale if not already
    transforms.RandomResizedCrop(size =256, scale= (0.8, 1.0) ),
    transforms.RandomRotation(degrees=15),
    transforms.RandomHorizontalFlip(),
```

```
        transforms.CenterCrop(size =224),
        transforms.ToTensor(),  # Convert images to PyTorch tensors
        transforms.Normalize(mean=[0.5], std=[0.5])  # Normalize the images
    ])
```

Splitting the dataset:

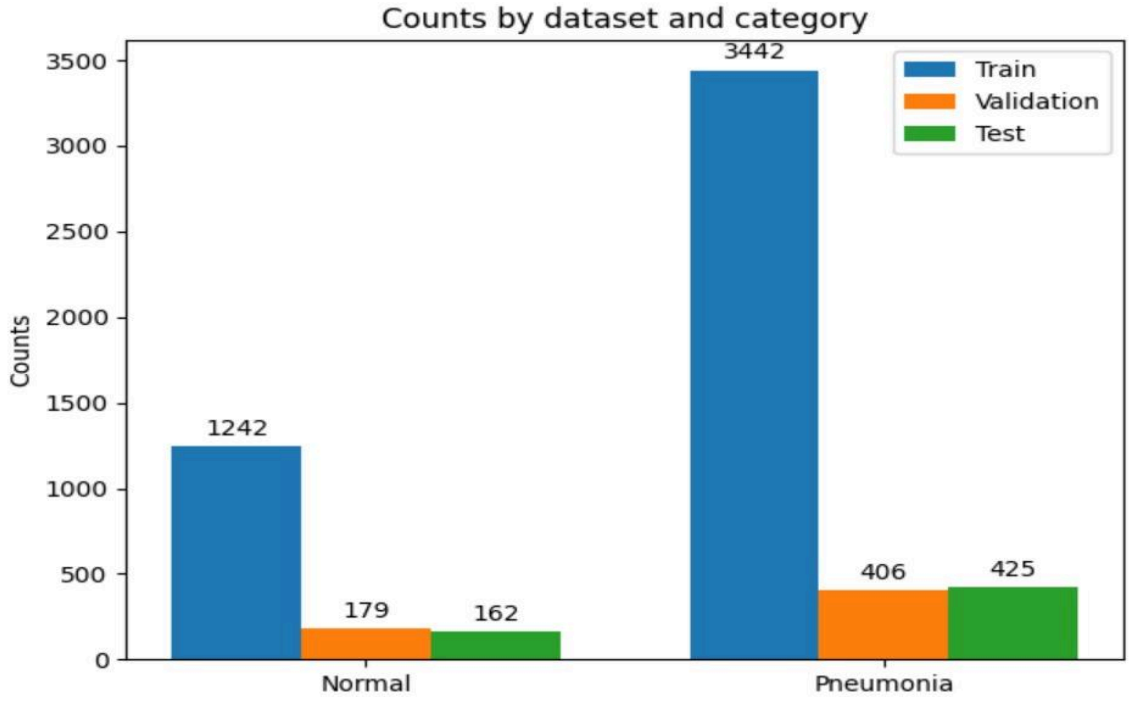
The data is split into 3 parts with total size as total length of dataset.

Trainsize= 0.80* Total size

valid_size= 0.10* Total size

test_size= 0.10*Total size

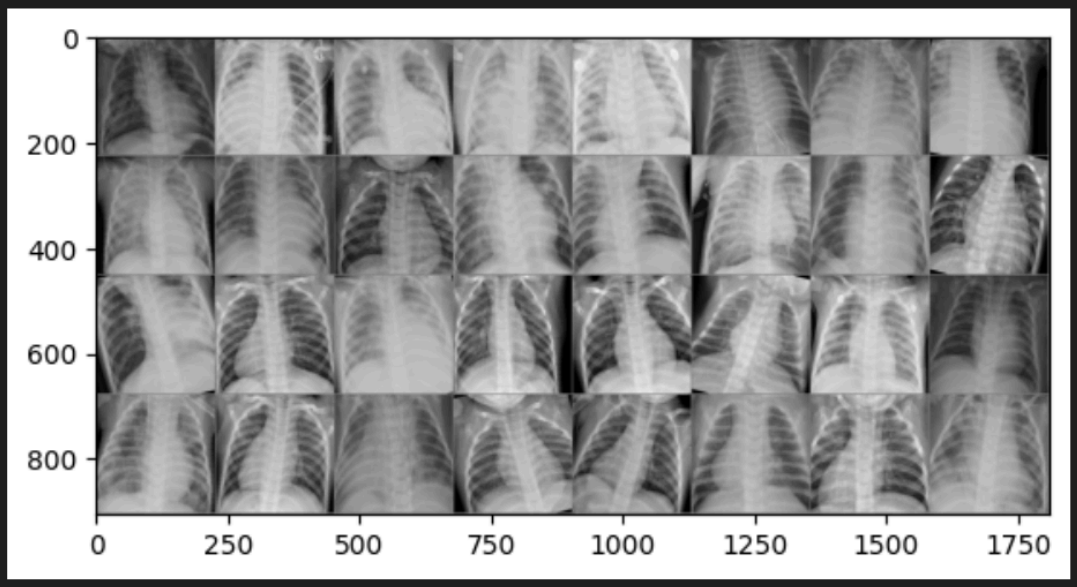
The training dataset consists of 1242 images of Normal and 3442 images of Pneumonia. The Validation dataset consists of 179 images of Normal and 406 images of Pneumonia. The testing dataset consists of 162 images of Normal and 425 images of Pneumonia. Below is the graph showing the distribution of Normal and Pneumonia images into train, validation and test data



Visualization:

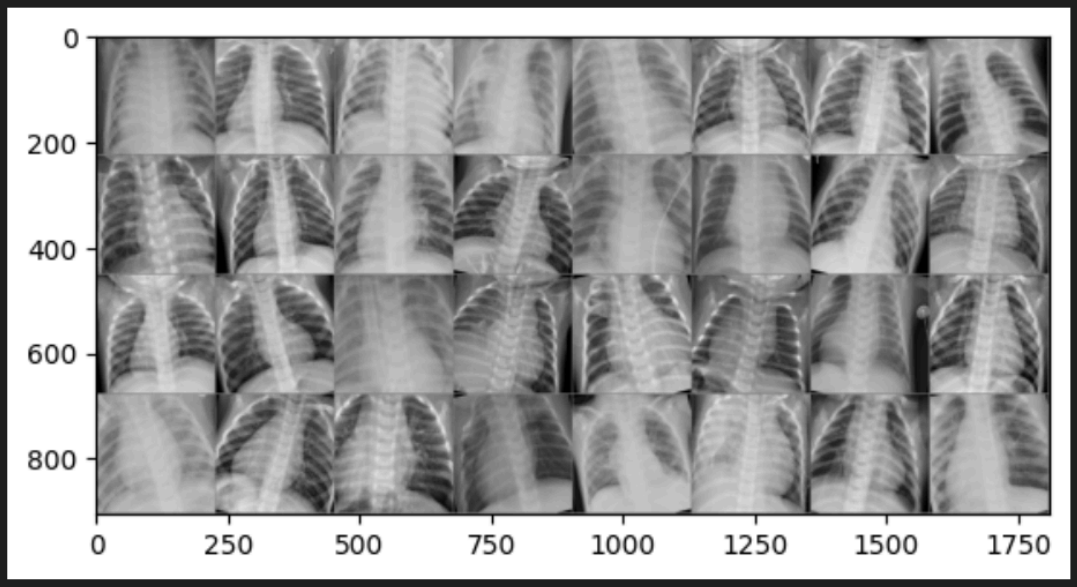
Below are the visualizations of the x-ray images after performing transformations and splitting the data into train, validation and test dataset.

Train Set:

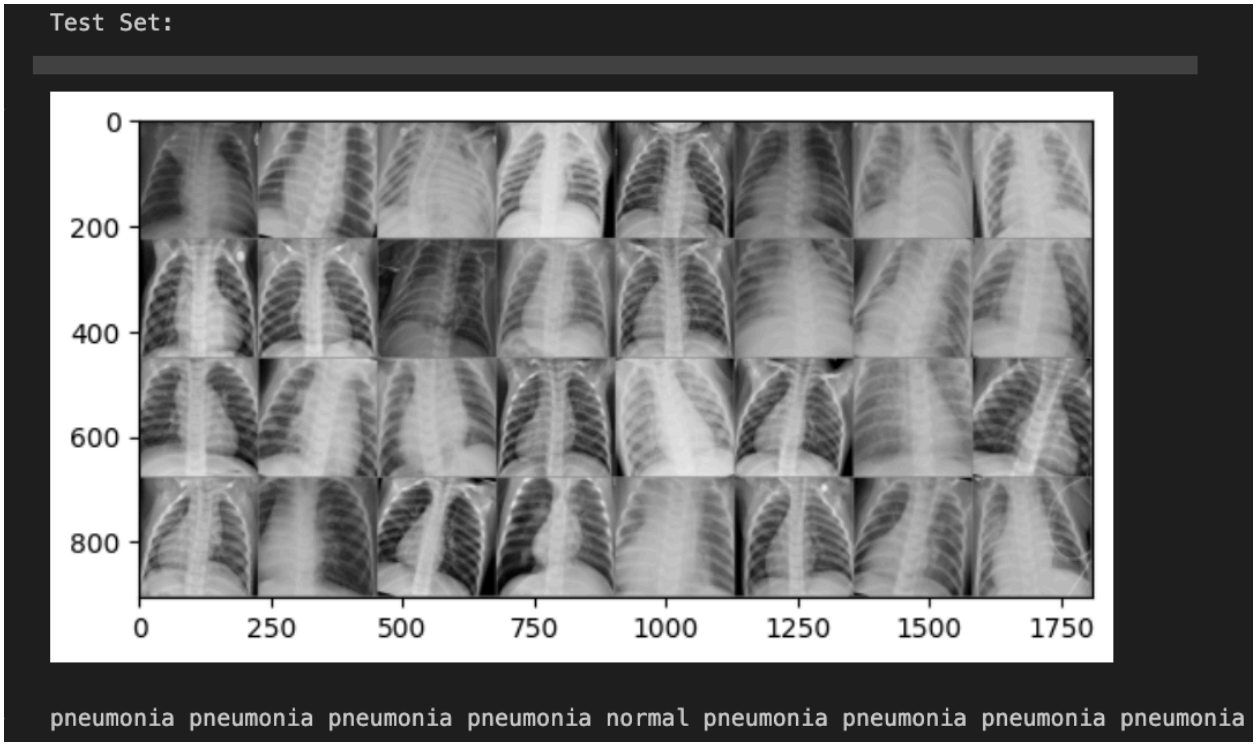


pneumonia pneumonia pneumonia pneumonia pneumonia pneumonia pneumonia pneumonia pneumor

Validation Set:



pneumonia normal pneumonia pneumonia pneumonia normal normal pneumonia pneumonia normal



CNN Model Evaluation:
1) Base CNN:

The `Base_CNN` class has three convolutional layers `conv1`, `conv2` and `conv3` with filter counts 64, 128, 256 for feature extraction. Each layer is followed by a ReLU activation function and a max pooling layer that splits the spatial dimensions into 2.

After the convolutional layers, the network flattens the output and passes it through three fully connected layers: `fc1` with 1024 units, `fc2` with 512 units, and `fc3` that outputs the scores.

The `forward` method processes input images through convolutional layers with ReLU activation, pooling, and dropout. It flattens the output, and passes it through fully connected layers to produce class scores.

```
Base_CNN(
  (conv1): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=200704, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=2, bias=True)
)
```

On training the model,

The loss function (`CrossEntropyLoss`), and optimizer (`SGD` with learning rate 0.01 and momentum 0.9) are initialized. Validation is done without gradient computation to evaluate the model's performance.

For every epoch, we're finding the training loss, training accuracy, validation loss, validation accuracy for the Base CNN model.

Results:

```
Training on: cuda:0
Epoch 1/10, Loss: 0.4323, Training Accuracy: 80.25%
Validation Loss: 0.2861, Validation Accuracy: 88.21%
Epoch 2/10, Loss: 0.2585, Training Accuracy: 89.15%
Validation Loss: 0.2657, Validation Accuracy: 90.09%
Epoch 3/10, Loss: 0.2217, Training Accuracy: 91.50%
Validation Loss: 0.2256, Validation Accuracy: 90.09%
Epoch 4/10, Loss: 0.2127, Training Accuracy: 92.10%
Validation Loss: 0.1852, Validation Accuracy: 93.16%
Epoch 5/10, Loss: 0.1912, Training Accuracy: 92.66%
Validation Loss: 0.1833, Validation Accuracy: 92.99%
Epoch 6/10, Loss: 0.1958, Training Accuracy: 92.63%
Validation Loss: 0.2052, Validation Accuracy: 93.33%
Epoch 7/10, Loss: 0.1833, Training Accuracy: 92.81%
Validation Loss: 0.1833, Validation Accuracy: 94.36%
Epoch 8/10, Loss: 0.1718, Training Accuracy: 93.83%
Validation Loss: 0.1650, Validation Accuracy: 94.87%
Epoch 9/10, Loss: 0.1637, Training Accuracy: 94.02%
Validation Loss: 0.1714, Validation Accuracy: 93.68%
Epoch 10/10, Loss: 0.1673, Training Accuracy: 93.60%
Validation Loss: 0.2214, Validation Accuracy: 91.45%
```

2) CNN with dropout layers and early stopping:

The `Adv_CNN` class consists of three convolutional layers `conv1`, `conv2`, `conv3` with filter counts increasing 64, 128, 256 and max pooling to reduce spatial dimensions.

Dropout layers are applied with 50% drop rate after each convolutional layer to avoid overfitting. The network has three fully connected layers (`fc1`, `fc2`, `fc3`), with dropout applied after the first two.

The `forward` method processes input images through convolutional layers with ReLU activation, pooling, and dropout. It flattens the output, and passes it through fully connected layers to produce class scores.

```
Adv_CNN(
  (conv1): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout_conv): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=200704, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=2, bias=True)
  (dropout_fc): Dropout(p=0.5, inplace=False)
)
```

The loss function (`CrossEntropyLoss`), and optimizer (`SGD` with learning rate 0.01 and momentum 0.9) are initialized.

Validation is done with gradient computation to evaluate the model's performance. We're using early stopping to prevent overfitting of the model.

For every epoch, we're finding the training loss, training accuracy, validation loss, validation accuracy for the Adv_CNN model.

Results:

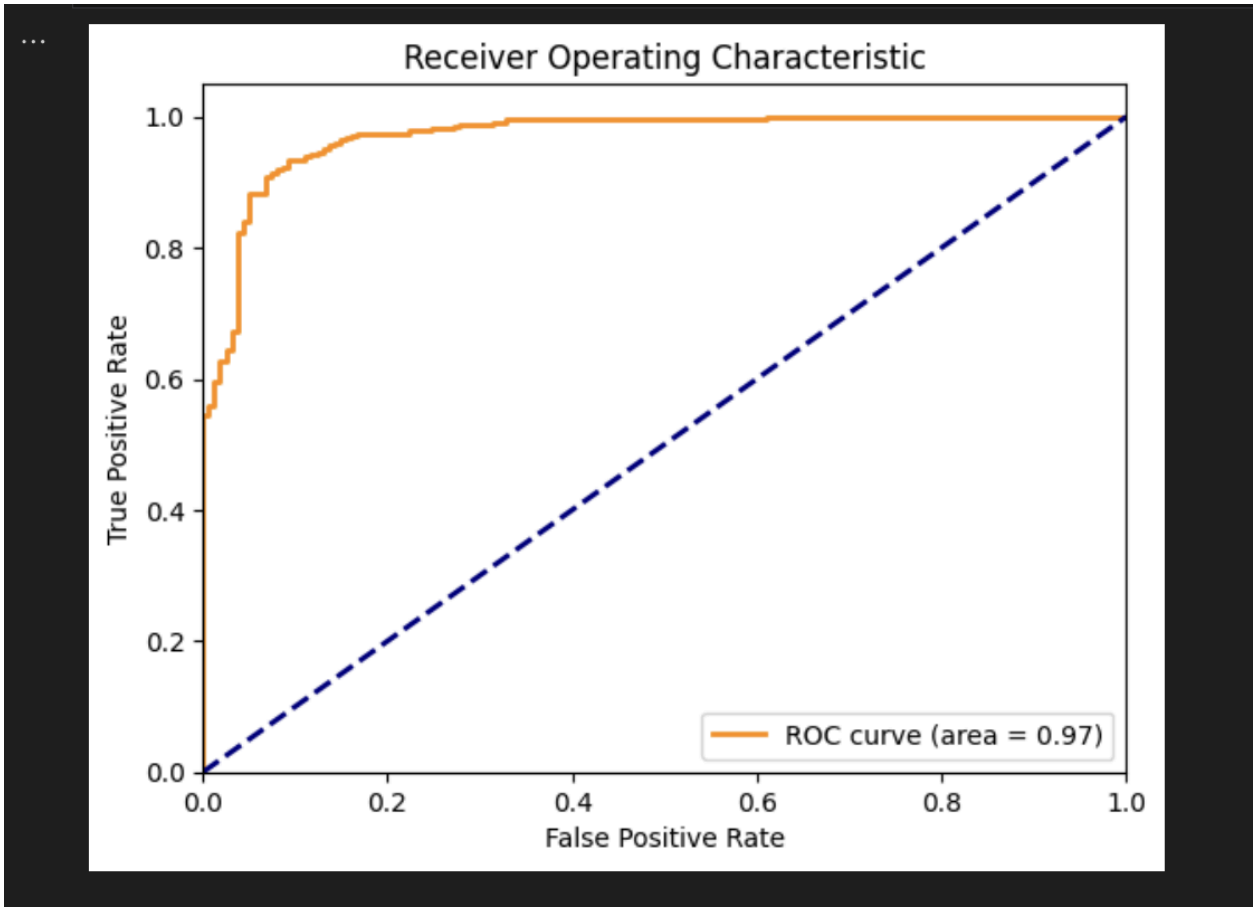
```
Training on: cuda:0
Epoch 1/15, Loss: 0.4881, Training Accuracy: 77.26%
Validation Loss: 0.4104, Validation Accuracy: 84.96%
Epoch 2/15, Loss: 0.3248, Training Accuracy: 86.29%
Validation Loss: 0.3051, Validation Accuracy: 90.77%
Epoch 3/15, Loss: 0.2943, Training Accuracy: 88.81%
Validation Loss: 0.3575, Validation Accuracy: 83.59%
Epoch 4/15, Loss: 0.2486, Training Accuracy: 90.33%
Validation Loss: 0.2848, Validation Accuracy: 88.38%
Epoch 5/15, Loss: 0.2379, Training Accuracy: 90.69%
Validation Loss: 0.3548, Validation Accuracy: 84.96%
Epoch 6/15, Loss: 0.2160, Training Accuracy: 91.52%
Validation Loss: 0.2217, Validation Accuracy: 92.65%
Epoch 7/15, Loss: 0.2172, Training Accuracy: 91.84%
Validation Loss: 0.2120, Validation Accuracy: 92.82%
Epoch 8/15, Loss: 0.2101, Training Accuracy: 91.99%
Validation Loss: 0.2177, Validation Accuracy: 92.48%
Epoch 9/15, Loss: 0.2059, Training Accuracy: 92.06%
Validation Loss: 0.2151, Validation Accuracy: 93.16%
Early stopping at epoch 9 out of 15
Saved best model with validation loss: 0.2120

<All keys matched successfully>
```

The performance metrics and analyze the results:

We're calculating various metrics like accuracy, precision, recall, and F1 score. It performs inference on the test data without gradient calculation.

```
Test Accuracy: 92.84%
Test Accuracy: 92.84%
Final Precision: 0.9284
Final Recall: 0.9284
Final F1 Score: 0.9284
```

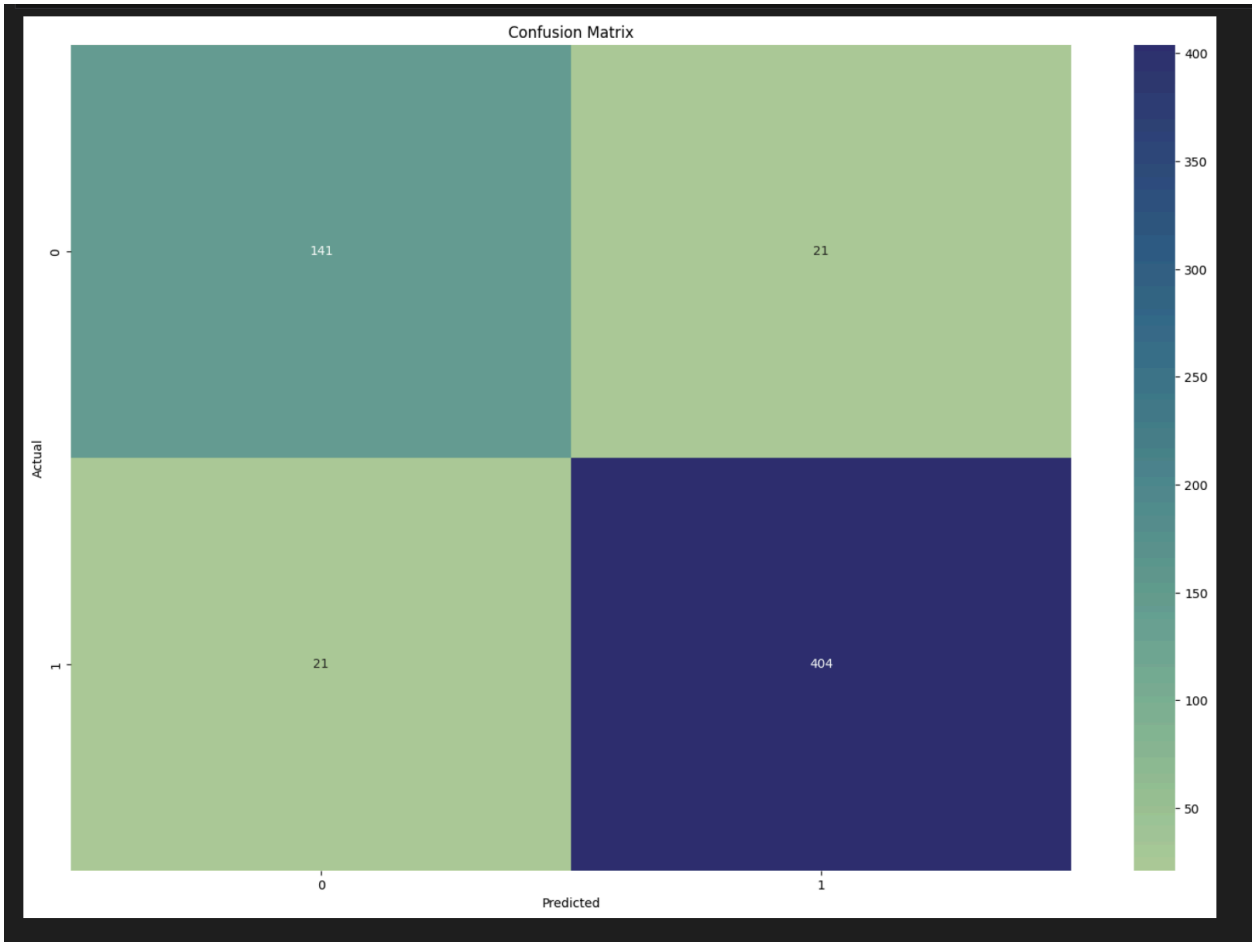
Visualizations of the Metrics:
Receiver Operating Characteristic(ROC):



The ROC of true labels is calculated which contains an ROC curve which displays the false positive rate with the true positive rate.

Confusion Matrix:

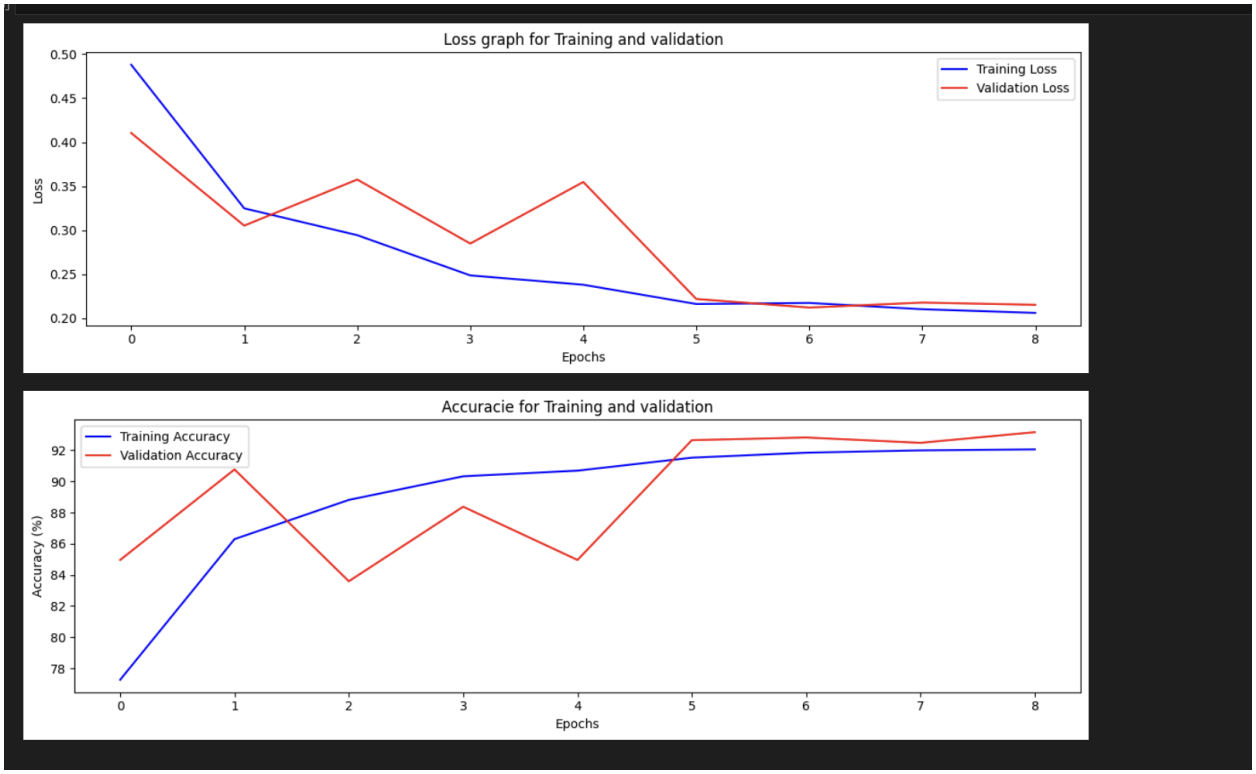
Each cell in the Confusion matrix heatmap represents the count of true positives, false positives, false negatives, and true negatives.
The x-axis represents predicted labels and the y-axis represents actual labels.
The color intensity shows the count of samples for each class pair. This helps us in understanding the model's performance in classifying different categories.



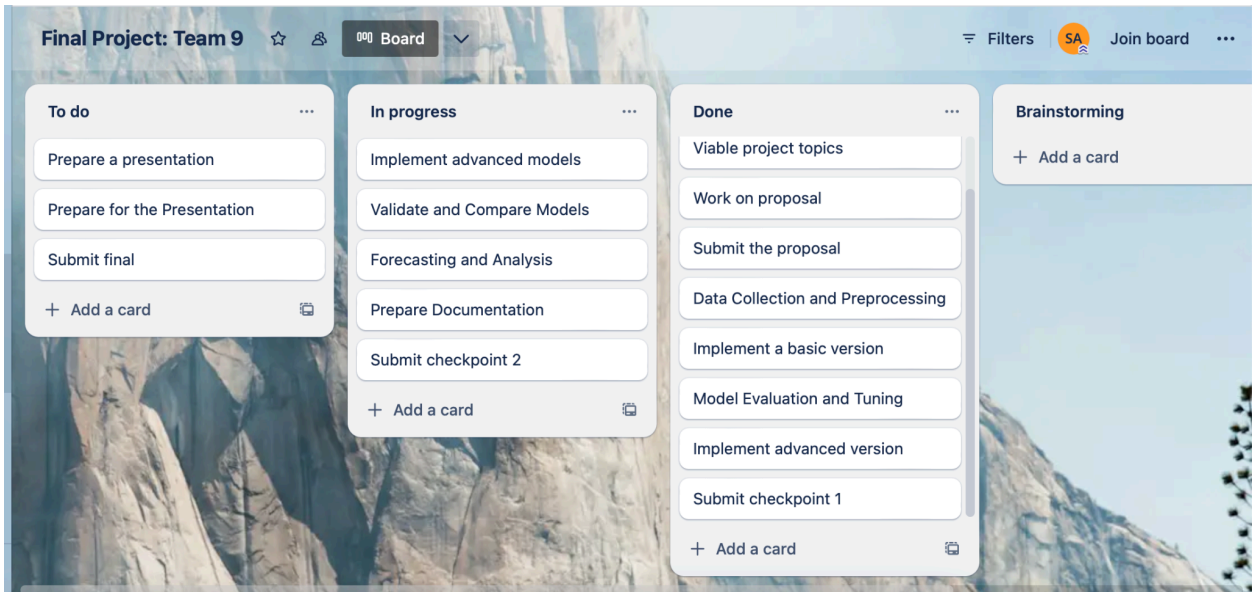
Loss and Accuracy Plotting:

The first plot shows the training loss (in blue) and validation loss (in red) over epochs.As training progresses, both losses ideally should decrease, indicating improved model performance.

The second plot illustrates the training accuracy (in blue) and validation accuracy (in red) over epochs. It provides insight into how well the model is performing on both training and validation sets.



Trello dashboard:



References:

1. https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia/data
2. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
3. https://pytorch.org/vision/stable/transforms.html
4. https://anushsom.medium.com/image-augmentation-for-creating-datasets-using-pytorch-for-dummies-by-a-dummy-a7c2b08c5bcb
5. https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/

CSE 676: Deep Learning, Summer 2024

Instructor: Alina Vereshchaka

Checkpoint - 2 Report

Team Members
Hariharan Sriram (hsriram)
Swanith Ambadas (swanitha)
Manish Bikumalla (manishbi)

Introduction:

The aim of this project is to leverage deep learning models like CNN and apply them to chest X-rays image data to classify whether a person suffers from pneumonia or not. As part of this checkpoint submission, we implemented 4 advanced CNN architectures and performed a comparative analysis of these models with the earlier applied basic version of CNN and its improved form. The findings are reported in this submission.

Steps covered in checkpoint 1:

We performed data processing and cleaning to the dataset. The data is split into training, testing and validation. Then applied CNN model and Improved CNN model to the dataset and trained the model to find the evaluation metrics such as Accuracy and Loss for training and testing and validate them over epochs.

Findings of checkpoint 1:

For the improved version of the CNN, the accuracy came out to be 92.84%. Based on other charts and performance metrics, the model seemed to have performed really well.

Checkpoint 2:

We're applying 4 complex CNN models to the dataset and evaluating the model based on metrics like accuracy, loss and visualizing the model's performance by plotting necessary graphs.

Models applied are:

1) VGG-19
2) DenseNet
3) GoogLeNet
4) ResNet

Model 1: VGG-19:

VGG19 is a convolutional neural network (CNN) model. It has 19 layers in total, 16 convolutional layers and 3 fully connected layers. Each convolutional layer has small

3x3 filters, 5 max-pooling layers. VGG19 uses max-pooling to reduce spatial dimensions and ReLU activation function for non-linearity. It is widely used for image recognition for its effective and simple design. It helps us achieve high performance and this makes it a powerful model for various applications.

Base VGG-19 Architecture:

The nn.Conv2d(1, 64, kernel_size=3, padding=1) changes the input to grayscale.
It has 64 o/p channels with 3x3 kernels with ReLU activation and Max Pooling.
Conv layer 3 has 64 i/p and 124 o/p channels with 3x3 kernels and a padding followed by ReLU and Max Pooling.

Conv Layer-5 has 128 i/p and 256 o/p channels with a 3x3 kernel and padding=1. It has a ReLU activation.

Conv Layer-9 has 256 i/p and 512 o/p channels with a 3x3 kernel and padding=1. It has a ReLU activation.

Conv Layer-13 has 512 i/p and 512 o/p channels with a 3x3 kernel and padding=1. It has a ReLU activation.

A total of 16 convolutional Layers is used in the VGG-19 model. The model has 3 fully connected layers each with ReLU activation function followed by Dropout.

The forward pass, the data flows properly through the layers and the output is flattened to 1.

Evaluation of the model:

On training and evaluating the model we find the following as the output for each epoch

```
Epoch [1/10] Training Loss: 0.5789, Training Accuracy: 74.90%
Epoch [1/10] Validation Loss: 0.8024, Validation Accuracy: 37.13%
Epoch [2/10] Training Loss: 0.4508, Training Accuracy: 80.36%
Epoch [2/10] Validation Loss: 0.4493, Validation Accuracy: 81.66%
Epoch [3/10] Training Loss: 0.3235, Training Accuracy: 85.78%
Epoch [3/10] Validation Loss: 0.6289, Validation Accuracy: 74.72%
Epoch [4/10] Training Loss: 0.3074, Training Accuracy: 88.12%
Epoch [4/10] Validation Loss: 0.4511, Validation Accuracy: 82.69%
Epoch [5/10] Training Loss: 0.3067, Training Accuracy: 87.36%
Epoch [5/10] Validation Loss: 0.6519, Validation Accuracy: 81.44%
Epoch [6/10] Training Loss: 0.2719, Training Accuracy: 89.12%
Epoch [6/10] Validation Loss: 0.4425, Validation Accuracy: 81.78%
Epoch [7/10] Training Loss: 0.2528, Training Accuracy: 90.39%
Epoch [7/10] Validation Loss: 1.0294, Validation Accuracy: 75.51%
Epoch [8/10] Training Loss: 0.2627, Training Accuracy: 89.80%
Epoch [8/10] Validation Loss: 0.2261, Validation Accuracy: 91.46%
Epoch [9/10] Training Loss: 0.2151, Training Accuracy: 91.07%
Epoch [9/10] Validation Loss: 0.2562, Validation Accuracy: 90.77%
Epoch [10/10] Training Loss: 0.2110, Training Accuracy: 91.73%
Epoch [10/10] Validation Loss: 0.1981, Validation Accuracy: 92.14%
Finished Training
Training time: 6899.79 seconds
```

We’re evaluating the performance of the model by calculating metrics like accuracy, recall, F1 score and Recall.

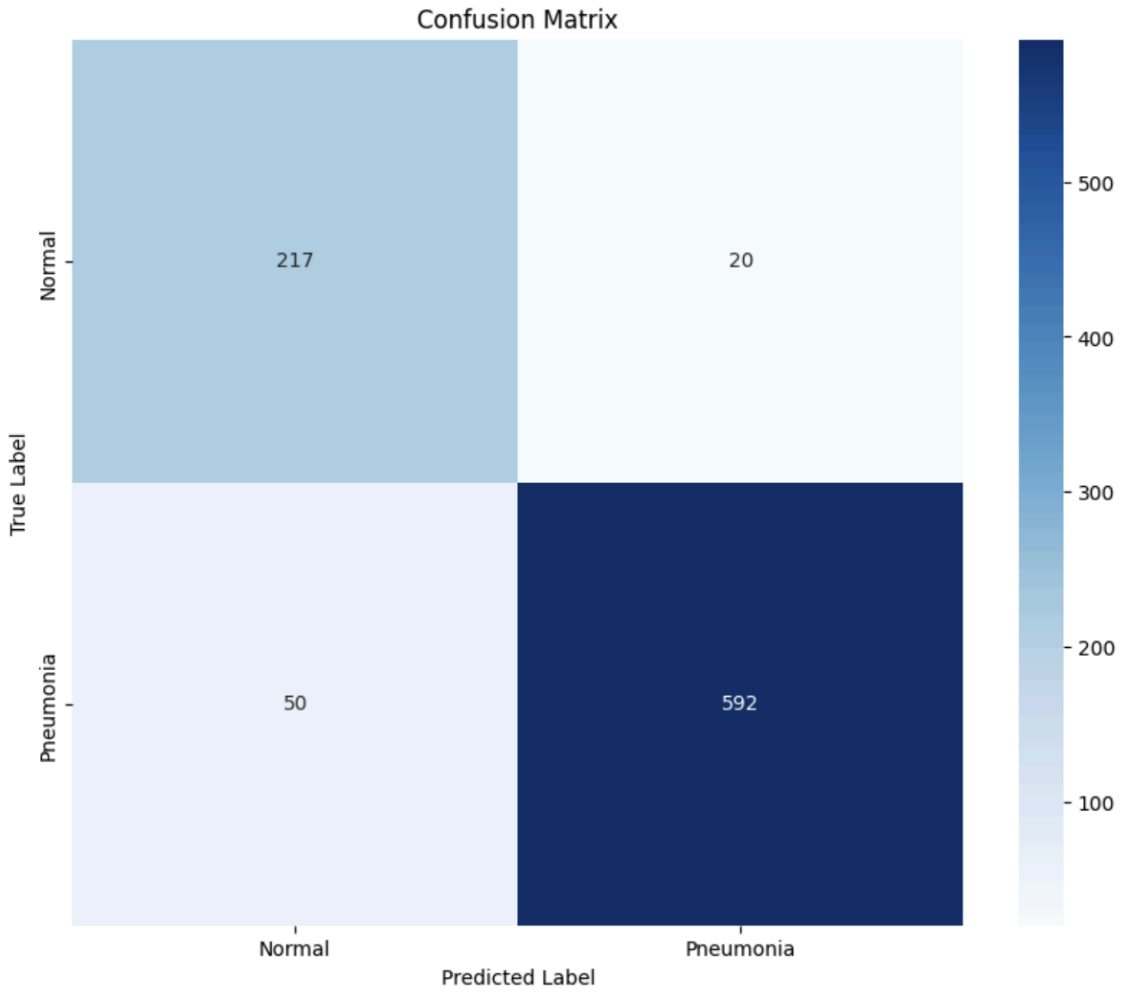
The below are the outputs of the model

Test Accuracy: 92.04%
Final Precision: 0.9256
Final Recall: 0.9204
Final F1 Score: 0.9218
Training time: 6899.79 seconds

Visualizations:

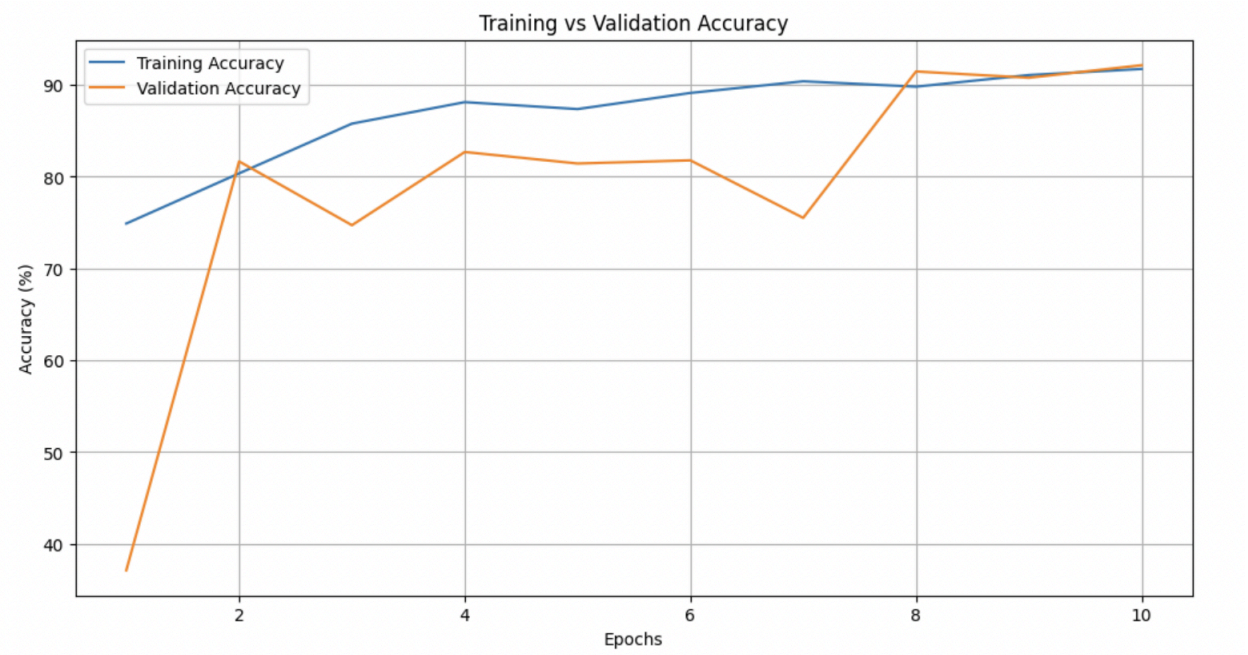
1) **Confusion matrix:**

   A confusion matrix is plotted from true labels and predictions. It is then visualized in the form of heatmap with axes for 'Normal' and 'Pneumonia' classes.

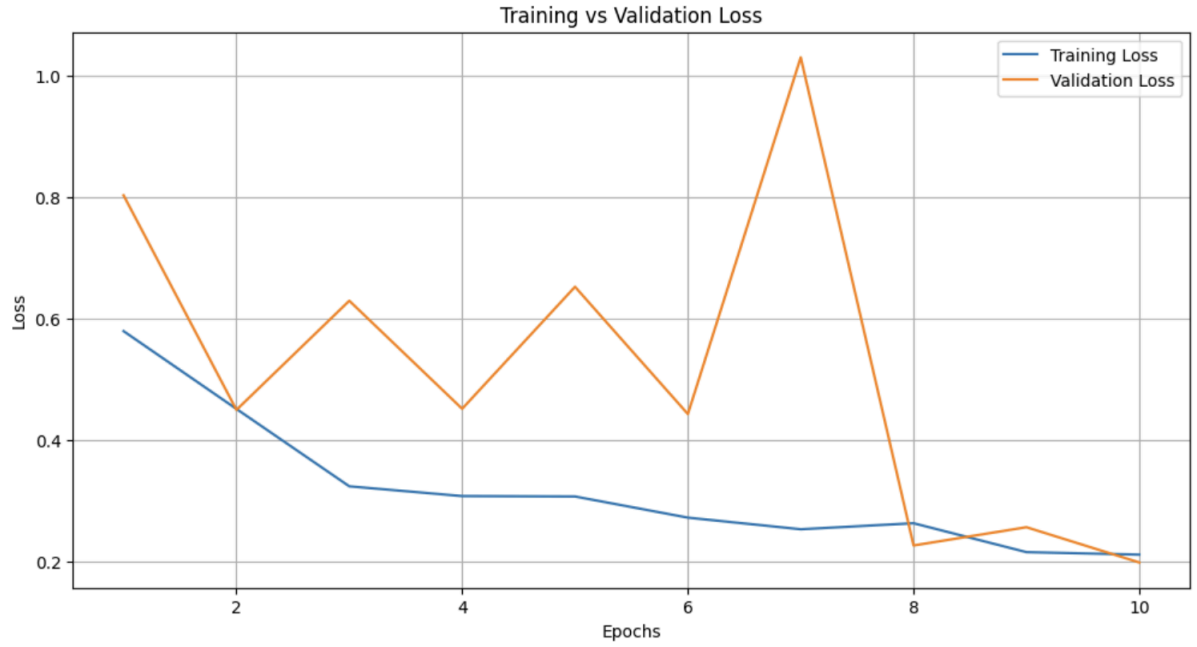


2) Accuracy and Loss:

It plots the training and validation of accuracies over epochs to differentiate between accuracies.

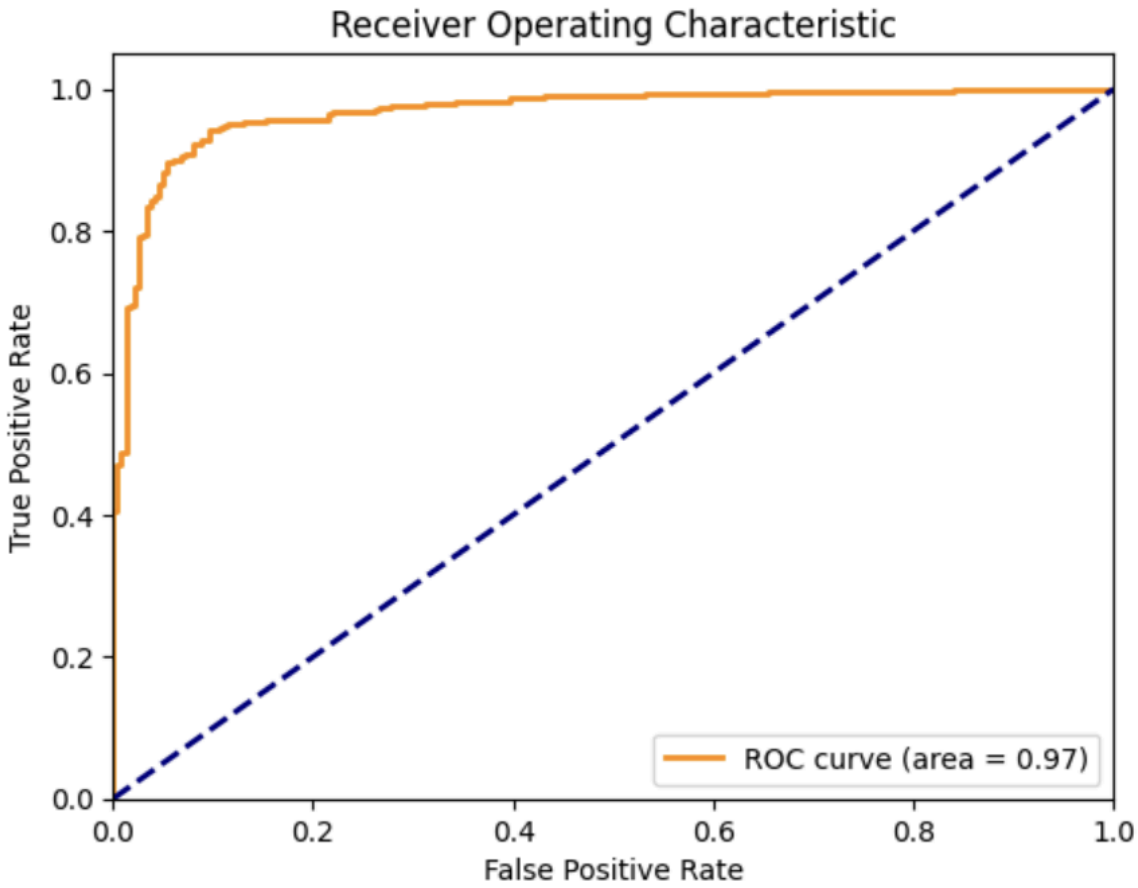


It plots the training and validation losses over each epoch to differentiate between losses.



3) Receiver Operating Characteristic

We're plotting the ROC curve for binary classification, showing the difference between true positive and false positive rates with an AUC value indicating model performance.



VGG 19 with dropout layers, weight decay and early stopping:

The updated VGG19 model has dropout layers. It introduces regularization by disabling units during training. Weight decay is used to add a penalty to the loss function to prevent overfitting. Early stopping monitors validation loss and stops training when performance reduces which optimizes model generalization and prevents overtraining.

We're checking if this is the best model and saving the best model.

```
Epoch [1/15] Training Loss Adv VGG19: 0.6201, Training Accuracy Adv VGG19: 71.72%
Epoch [1/15] Validation Loss Adv VGG19: 0.5486, Validation Accuracy Adv VGG19: 85.65%
Epoch [2/15] Training Loss Adv VGG19: 0.4805, Training Accuracy Adv VGG19: 78.97%
Epoch [2/15] Validation Loss Adv VGG19: 0.6058, Validation Accuracy Adv VGG19: 75.74%
Epoch [3/15] Training Loss Adv VGG19: 0.3696, Training Accuracy Adv VGG19: 84.09%
Epoch [3/15] Validation Loss Adv VGG19: 0.3536, Validation Accuracy Adv VGG19: 84.62%
Epoch [4/15] Training Loss Adv VGG19: 0.2974, Training Accuracy Adv VGG19: 87.12%
Epoch [4/15] Validation Loss Adv VGG19: 0.3459, Validation Accuracy Adv VGG19: 87.02%
Epoch [5/15] Training Loss Adv VGG19: 0.2935, Training Accuracy Adv VGG19: 87.92%
Epoch [5/15] Validation Loss Adv VGG19: 1.3452, Validation Accuracy Adv VGG19: 75.40%
Epoch [6/15] Training Loss Adv VGG19: 0.3518, Training Accuracy Adv VGG19: 86.63%
Epoch [6/15] Validation Loss Adv VGG19: 0.5988, Validation Accuracy Adv VGG19: 75.63%
Early stopping at epoch 6
Finished Training
Training time: 8024.14 seconds
Saved the best model with validation loss: 0.3459
```

We're evaluating the performance of the model by calculating metrics like accuracy, recall, F1 score and Recall.

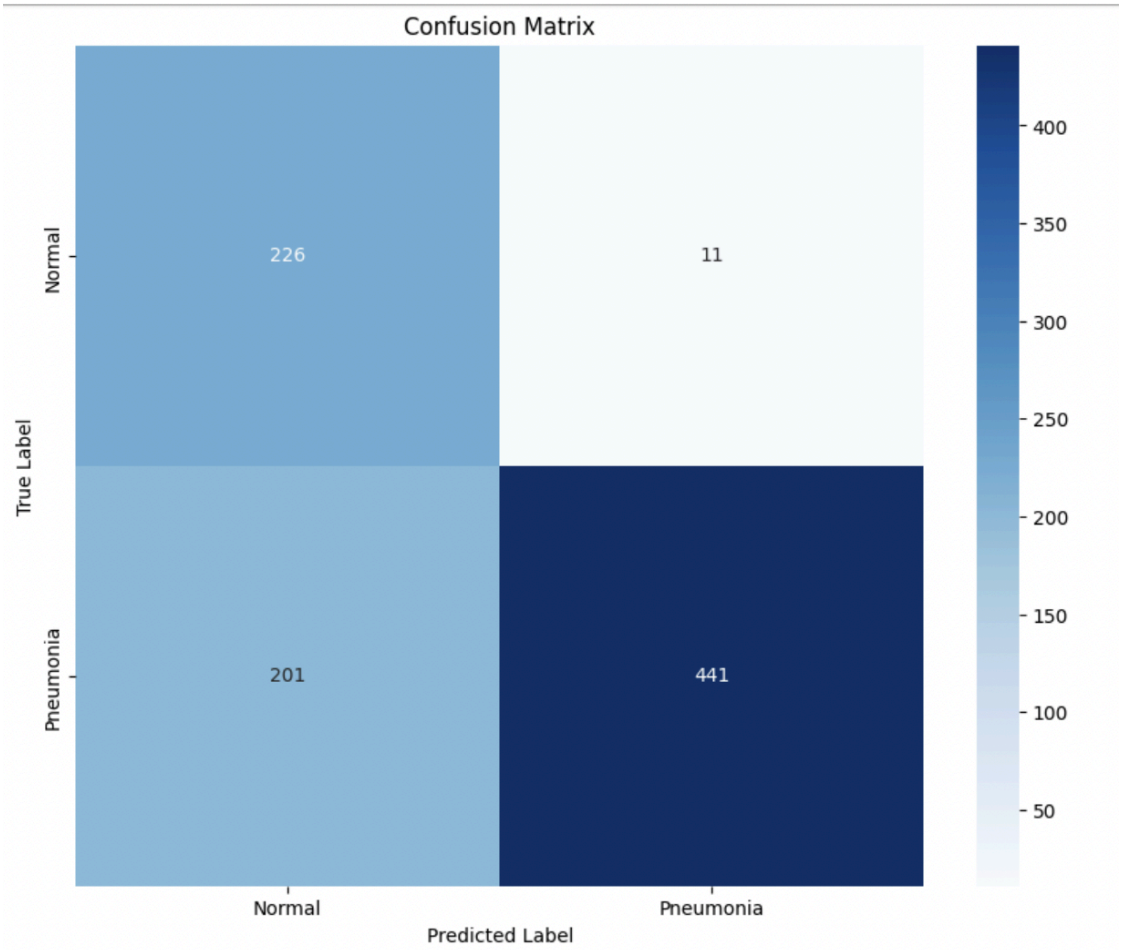
The below are the outputs of the model

Test Accuracy: 75.88%
Test Loss: 1.88
Final Precision: 0.9256
Final Recall: 0.9204
Final F1 Score: 0.9218

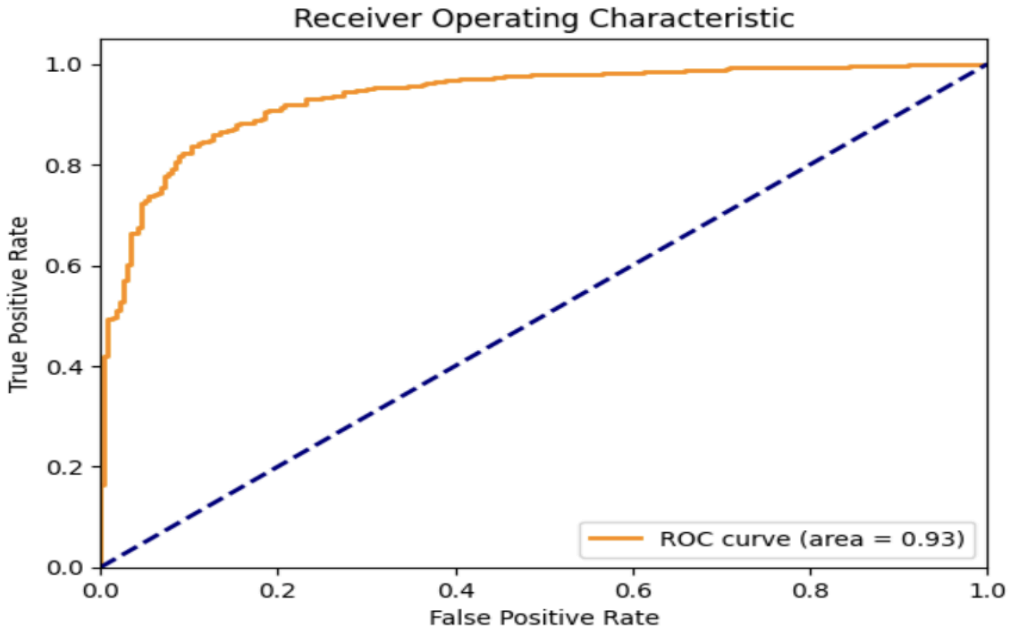
Visualizations:

1) **Accuracy and Loss:**
   We plot training and validation accuracies, and losses over epochs for an advanced VGG19 model. This shows the model's performance and effectiveness of regularization techniques like dropout and weight decay.

2) **Confusion Matrix:**
   A confusion matrix is plotted from true labels and predictions, then visualizes heatmap with labeled axes for 'Normal' and 'Pneumonia' classes.



3) Receiver Operating Characteristic:



Model 2: DenseNet

DenseNet(Densely Connected Convolutional Networks) is a CNN architecture known for its dense connections between layers. In this design, each layer receives input from all previous layers and sends its output to all subsequent layers. This structure improves gradient flow and promotes feature reuse, which helps mitigate the vanishing gradient problem.

Key Components:

- In dense block each layer receives inputs from the concatenated outputs of all preceding layers. This encourages feature reuse and minimizes the number of parameters.
- Located between dense blocks, transition layers reduce the number of feature maps and down-sample the spatial dimensions using convolution and average pooling.
- Growth rate indicates the number of feature maps added per dense layer

Process:

The network begins with a convolutional layer that adjusts the input channels to 1, followed by batch normalization, ReLU activation, and max pooling.

Each dense block contains multiple ‘Dense Layer’ instances, which include batch normalization, ReLU activation, 1x1 and 3x3 convolutions, and optional dropout.

The final section of the network consists of a batch normalization layer, followed by a global average pooling layer, and a fully connected layer (classifier) that generates predictions for the specified number of classes.

Evaluation:
On training and evaluating the model, we get the following outputs for each epoch.

```
Training on: cuda:0
Epoch 1/10, Loss: 0.3676, Training Accuracy: 83.80%
Validation Loss: 1.2832, Validation Accuracy: 79.50%
Epoch 2/10, Loss: 0.3548, Training Accuracy: 85.63%
Validation Loss: 0.4731, Validation Accuracy: 88.27%
Epoch 3/10, Loss: 0.2614, Training Accuracy: 89.56%
Validation Loss: 0.2607, Validation Accuracy: 89.98%
Epoch 4/10, Loss: 0.2471, Training Accuracy: 89.83%
Validation Loss: 0.4192, Validation Accuracy: 85.31%
Epoch 5/10, Loss: 0.2223, Training Accuracy: 91.53%
Validation Loss: 0.4339, Validation Accuracy: 86.56%
Epoch 6/10, Loss: 0.2196, Training Accuracy: 91.34%
Validation Loss: 0.2259, Validation Accuracy: 91.12%
Epoch 7/10, Loss: 0.2039, Training Accuracy: 92.53%
Validation Loss: 0.3353, Validation Accuracy: 89.07%
Epoch 8/10, Loss: 0.2778, Training Accuracy: 88.80%
Validation Loss: 0.5141, Validation Accuracy: 80.18%
Epoch 9/10, Loss: 0.2165, Training Accuracy: 91.46%
Validation Loss: 1.1345, Validation Accuracy: 70.05%
Epoch 10/10, Loss: 0.2088, Training Accuracy: 91.75%
Validation Loss: 0.2256, Validation Accuracy: 90.77%
```

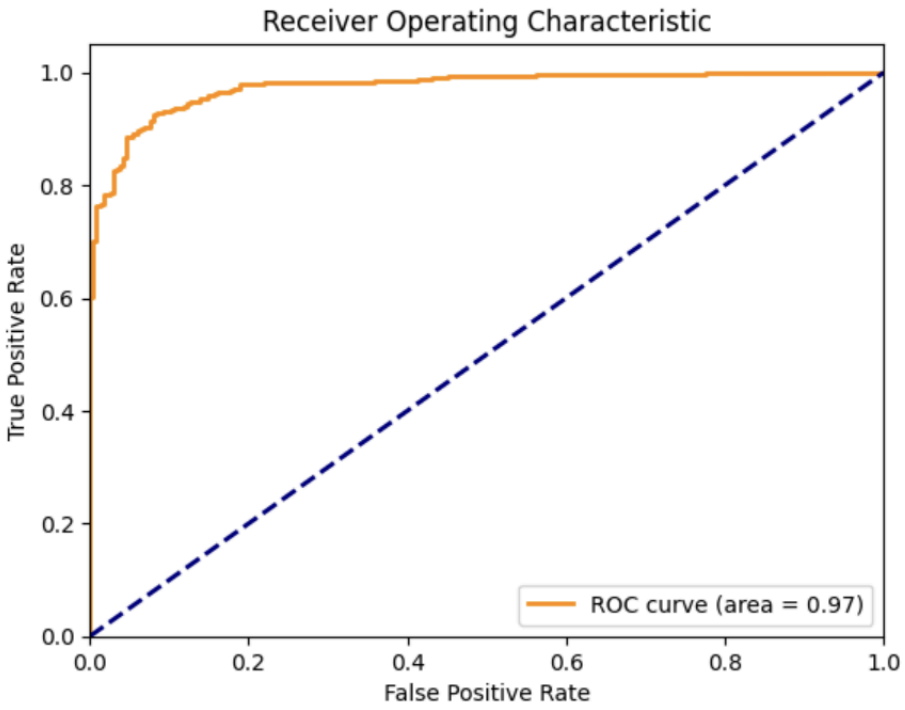
We're evaluating the performance of the model by calculating metrics like accuracy, recall, F1 score and Recall.

The below are the outputs of the model

Test Accuracy: 92.95%
Final Precision: 0.9298
Final Recall: 0.9295
Final F1 Score: 0.9275

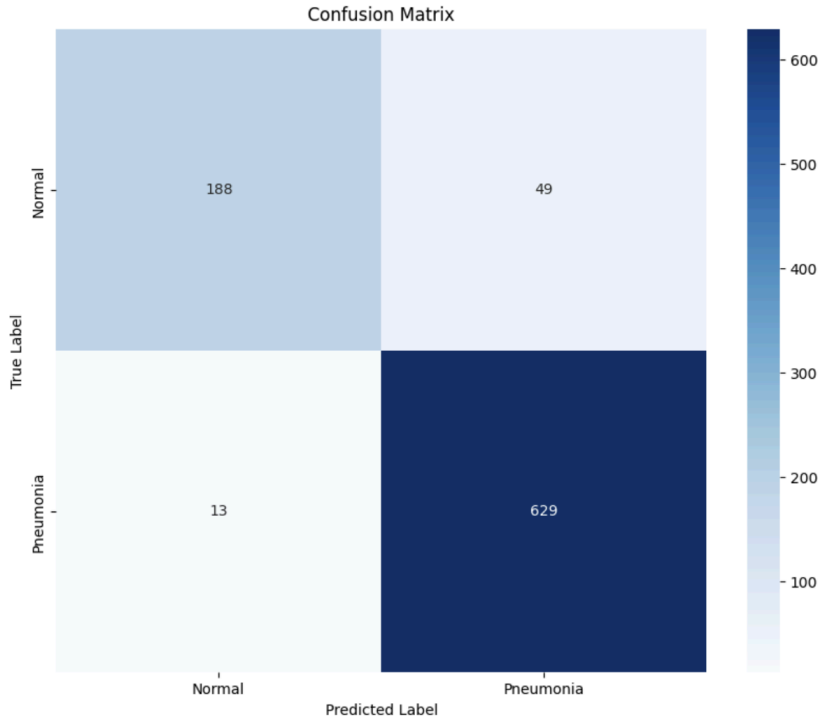
Visualizations:

1) **Receiver Operating Characteristic:**
   We're plotting the ROC curve for binary classification, showing the difference between true positive and false positive rates with an AUC value indicating model performance.



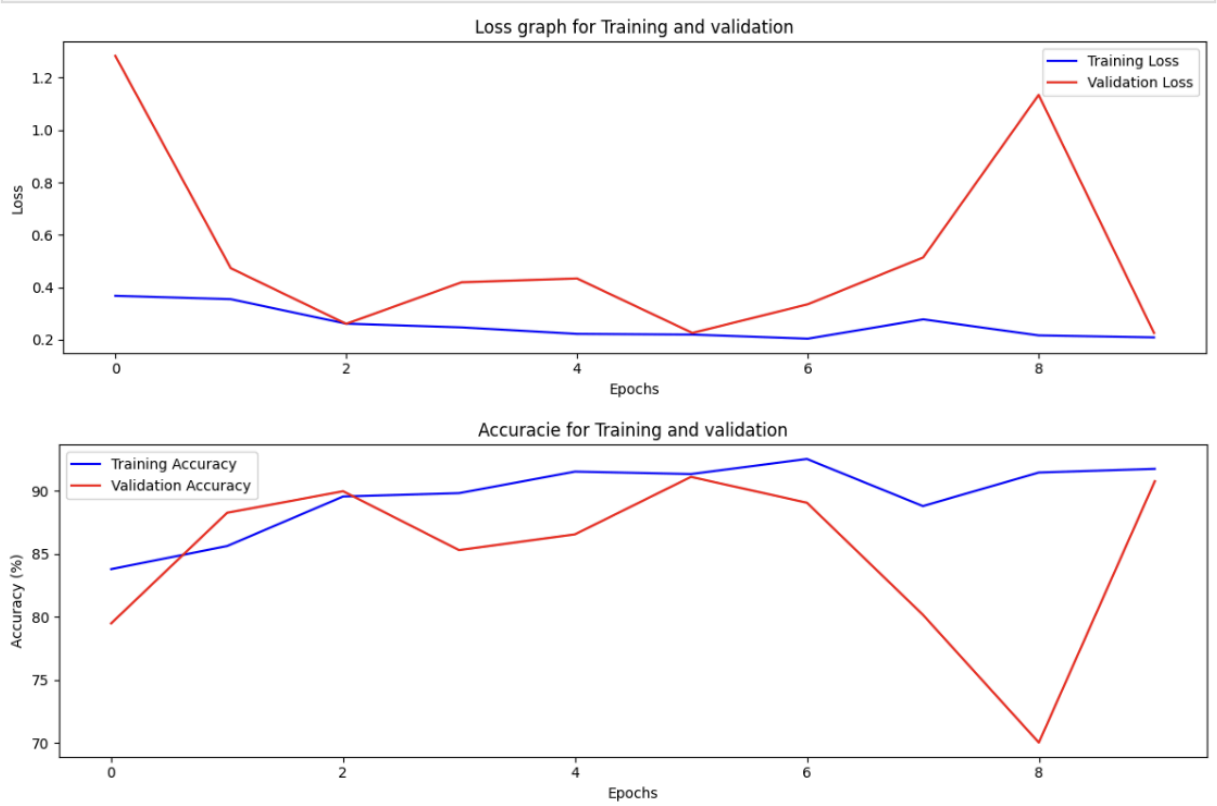
2) Confusion Matrix:

A confusion matrix is plotted from true labels and predictions, then visualizes heatmap with labeled axes for 'Normal' and 'Pneumonia' classes.



3) Accuracy and Loss:

Below are the loss and accuracy plots for training and validation data over the epochs for the DenseNet Model.



Improved DenseNet Model

The Improved DenseNet model has an addition of dropout layers. This deactivates neurons during training to prevent overfitting.
An L2 regularization is used to deal with large weights. This further prevents overfitting. Thus enhancing the overall performance of the model.

We're implementing early stopping for this and saving the weights of the best model.

On training and evaluating the model, we get the following outputs for each epoch.

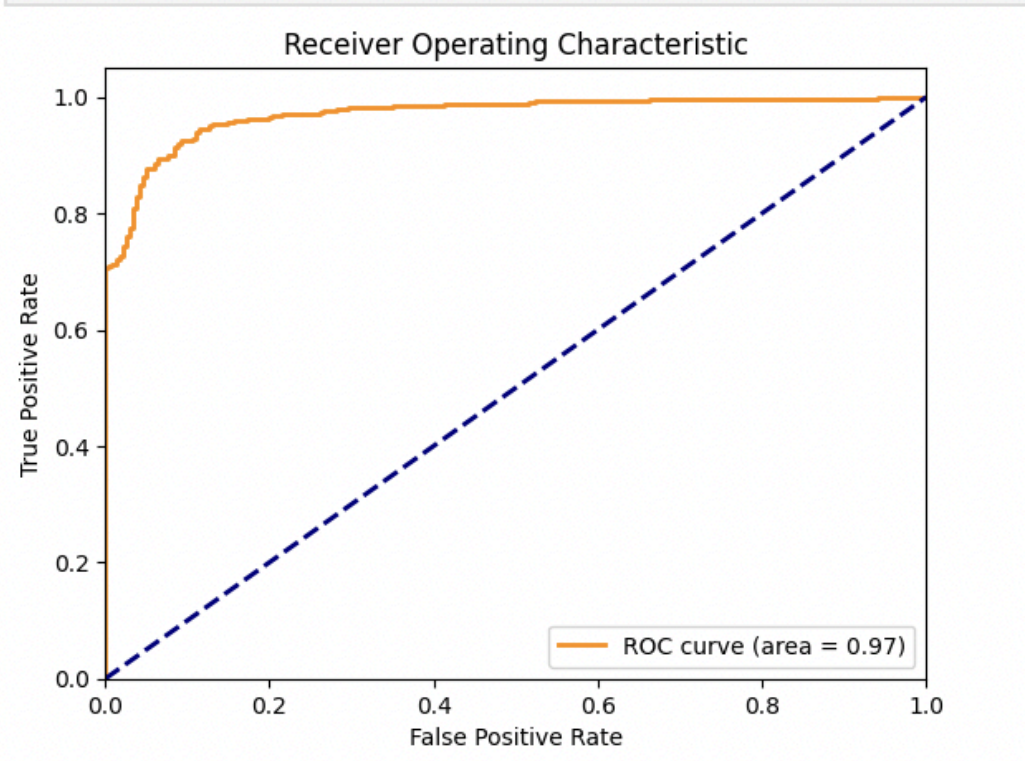
```
Training on: cuda:0
Epoch 1/15, Loss: 0.2909, Training Accuracy: 87.27%
Validation Loss: 0.7171, Validation Accuracy: 75.40%
Epoch 2/15, Loss: 0.2467, Training Accuracy: 90.10%
Validation Loss: 0.7997, Validation Accuracy: 75.40%
Epoch 3/15, Loss: 0.2350, Training Accuracy: 90.90%
Validation Loss: 0.2516, Validation Accuracy: 88.38%
Epoch 4/15, Loss: 0.2003, Training Accuracy: 92.46%
Validation Loss: 0.3542, Validation Accuracy: 86.56%
Epoch 5/15, Loss: 0.1944, Training Accuracy: 92.63%
Validation Loss: 0.4049, Validation Accuracy: 85.65%
Early stopping at epoch 5 out of 15
Saved best model with validation loss: 0.2516
<All keys matched successfully>
```

The below are the outputs for the evaluation metrics.

Test Accuracy: 92.04%
Final Precision: 0.9194
Final Recall: 0.9204
Final F1 Score: 0.9188

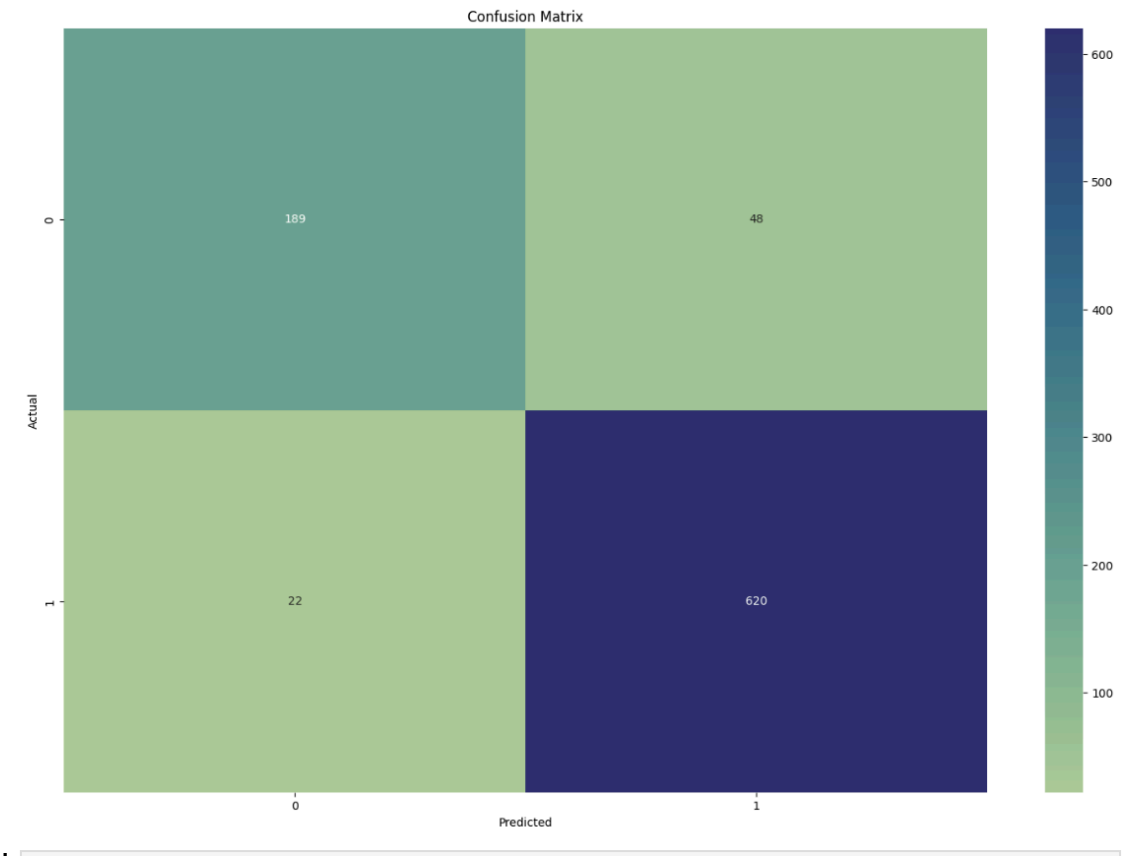
Visualizations:

1) **Receiver Operating Characteristic:**
   We're plotting the ROC curve for binary classification, showing the difference between true positive and false positive rates with an AUC value indicating model performance.



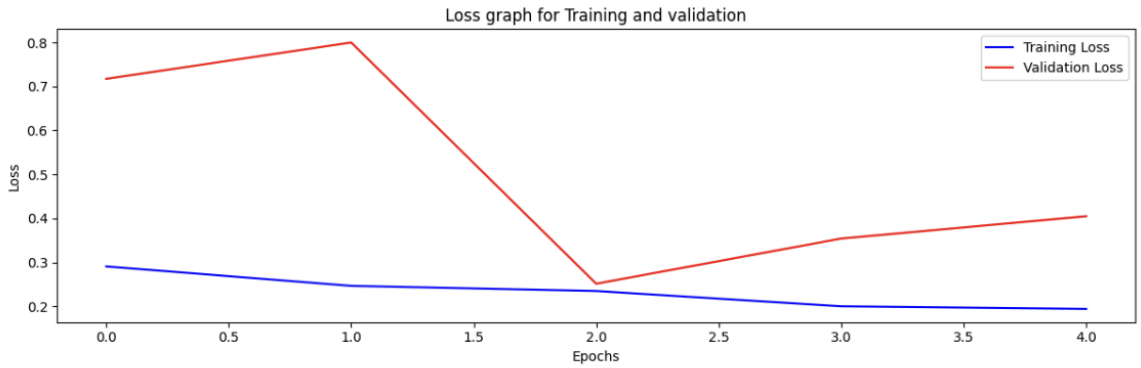
2) Confusion Matrix:

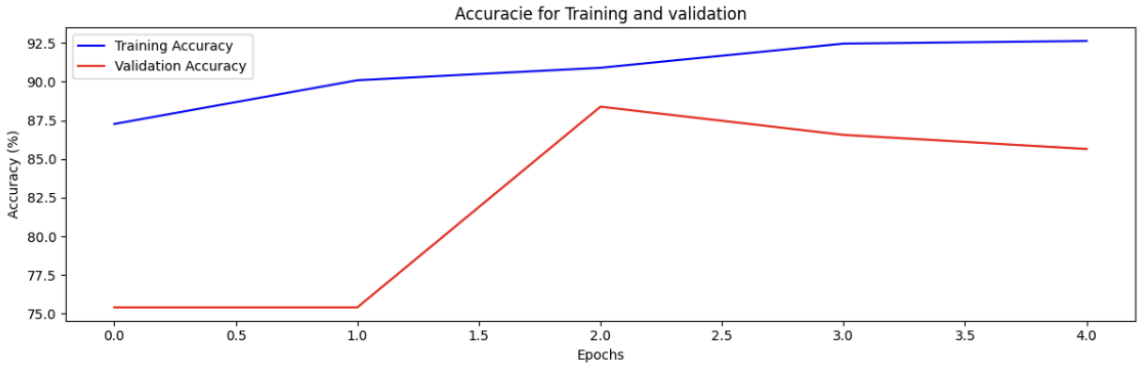
A confusion matrix is displayed with class labels ('Normal', 'Pneumonia') and visualizing it as a heatmap



3) Accuracy and Loss:

Training and validation losses and accuracies across epochs are plotted. Losses are represented in blue (training) and red (validation) and accuracies are shown which helps in performance evaluation.





Model 3: GoogleNet:

GoogLeNet is a convolutional neural network(CNN) that uses 1x1, 3x3, and 5x5 convolutions within the same layer to capture features more effectively. It uses global average pooling instead of fully connected layers. Due to this major advantage of the model, it is used for image classification, detection and increasing the efficiency and accuracy.

For this architecture a pre-trained model available in pytorch is being used.

```
import torchvision.models as models
weights = models.GoogLeNet_Weights.DEFAULT
model = models.googlenet(weights=weights)
model.conv1.conv = nn.Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
num_classes = 2
model.fc = nn.Linear(model.fc.in_features, num_classes)
```

The following are the outputs obtained for training and validation dataset:

```
Epoch 1/10, Loss: 0.2199, Training Accuracy: 91.22%
Validation Loss: 0.2906, Validation Accuracy: 88.27%
Epoch 2/10, Loss: 0.1754, Training Accuracy: 93.75%
Validation Loss: 0.1562, Validation Accuracy: 94.42%
Epoch 3/10, Loss: 0.1649, Training Accuracy: 93.54%
Validation Loss: 0.1331, Validation Accuracy: 95.10%
Epoch 4/10, Loss: 0.1351, Training Accuracy: 95.07%
Validation Loss: 0.1237, Validation Accuracy: 96.13%
Epoch 5/10, Loss: 0.1401, Training Accuracy: 95.05%
Validation Loss: 0.1421, Validation Accuracy: 94.99%
Epoch 6/10, Loss: 0.1284, Training Accuracy: 94.97%
Validation Loss: 0.1969, Validation Accuracy: 92.71%
Epoch 7/10, Loss: 0.1300, Training Accuracy: 95.27%
Validation Loss: 0.1858, Validation Accuracy: 93.17%
Epoch 8/10, Loss: 0.1110, Training Accuracy: 96.02%
Validation Loss: 0.2528, Validation Accuracy: 89.86%
Epoch 9/10, Loss: 0.1055, Training Accuracy: 95.95%
Validation Loss: 0.1140, Validation Accuracy: 96.58%
Epoch 10/10, Loss: 0.1267, Training Accuracy: 95.46%
Validation Loss: 0.1722, Validation Accuracy: 93.62%
```

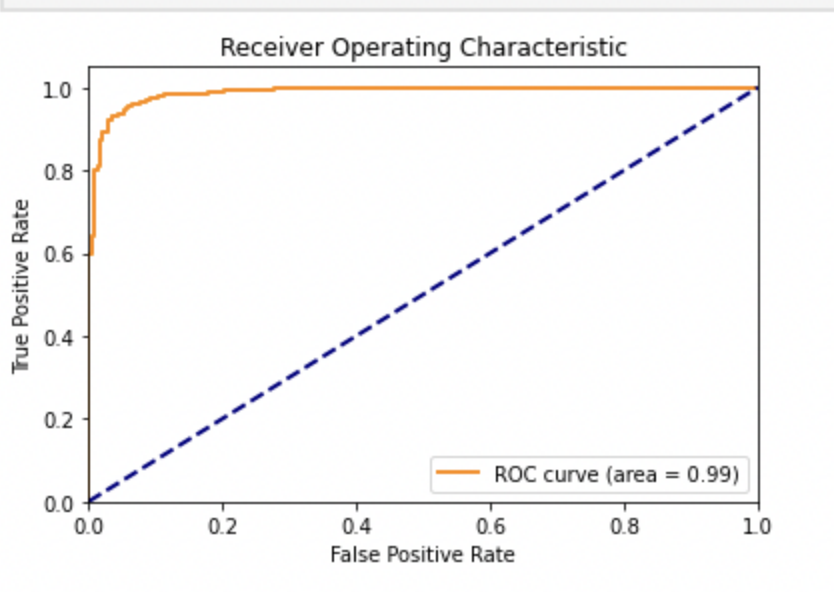
The below are the outputs for the evaluation metrics:

Test Accuracy: 95.22%
Final Precision: 0.9546
Final Recall: 0.9522
Final F1 Score: 0.9528

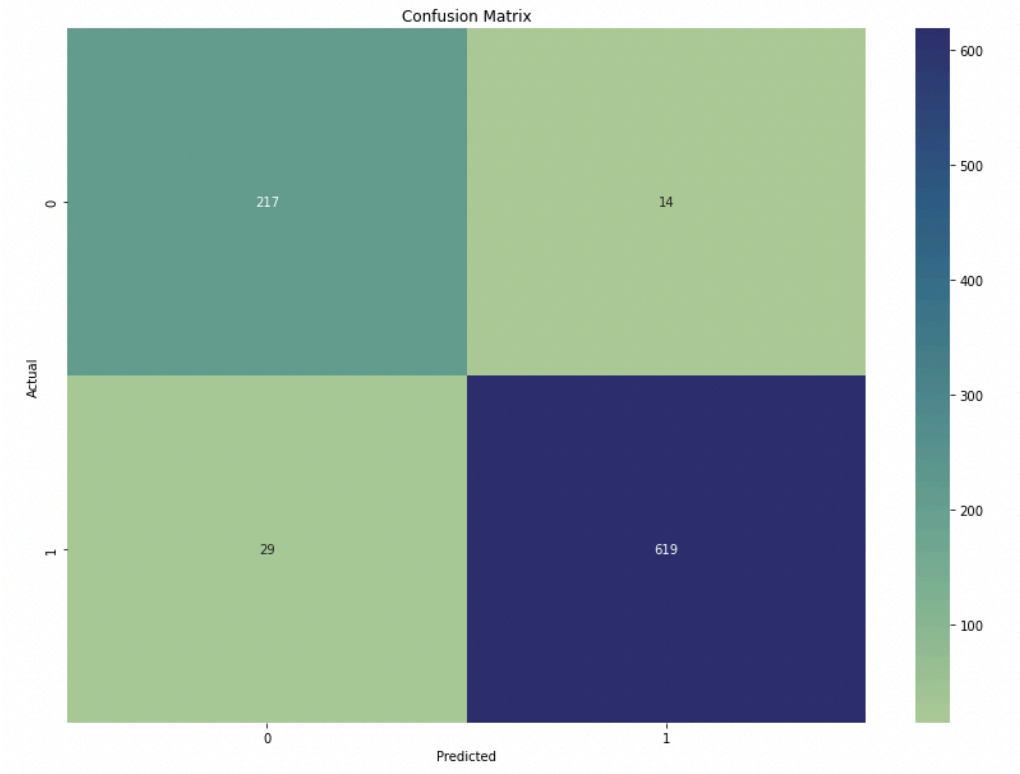
Visualization:

1) **Receiver Operating Characteristic:**

   We're plotting the ROC curve for binary classification, showing the difference between true positive and false positive rates with an AUC value indicating model performance.



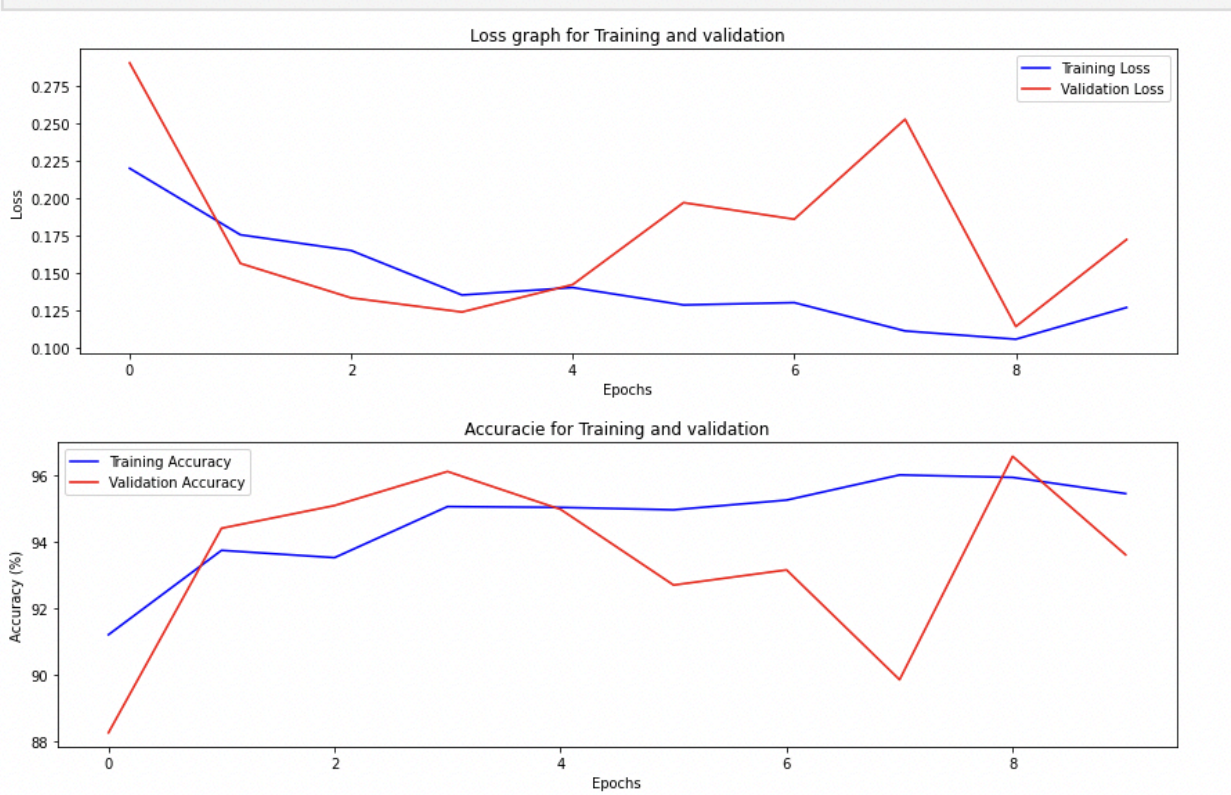
2) Confusion Matrix:



A confusion matrix is plotted with class labels ('Normal', 'Pneumonia') and visualizing it as a heatmap.

3) Accuracy and Loss:

Training and validation losses and accuracies across epochs are plotted.



Improved Googlenet Model

We're modifying a pre-trained GoogLeNet model for binary classification by adjusting its initial convolution layer, then adding a dropout layer before the final fully connected layer. An early stopping function and an L2 regularization in the form of weight decay is added to prevent overfitting of the model.

Training and Evaluation:

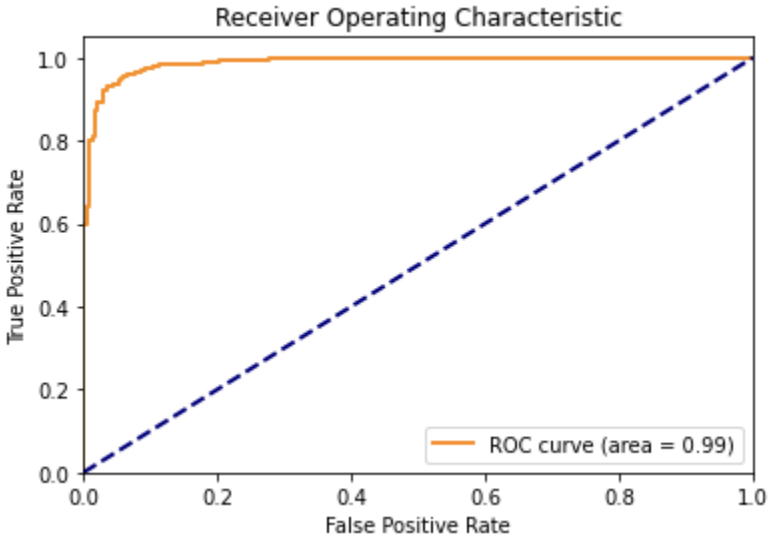
We're using a pre-trained model and evaluating it based on the training for each epoch. The following are the outputs. We're using early stopping.

```
model_imp.load_state_dict(torch.load('manishbi_hsriram_s
Epoch 1/15, Loss: 0.2348, Training Accuracy: 90.78%
Validation Loss: 0.2944, Validation Accuracy: 87.47%
Epoch 2/15, Loss: 0.1908, Training Accuracy: 92.78%
Validation Loss: 0.2623, Validation Accuracy: 88.50%
Epoch 3/15, Loss: 0.1613, Training Accuracy: 93.93%
Validation Loss: 0.1763, Validation Accuracy: 93.85%
Epoch 4/15, Loss: 0.1703, Training Accuracy: 93.46%
Validation Loss: 0.1797, Validation Accuracy: 92.03%
Epoch 5/15, Loss: 0.1575, Training Accuracy: 94.27%
Validation Loss: 0.1309, Validation Accuracy: 95.44%
Epoch 6/15, Loss: 0.1768, Training Accuracy: 93.27%
Validation Loss: 0.1761, Validation Accuracy: 92.60%
Epoch 7/15, Loss: 0.1523, Training Accuracy: 94.56%
Validation Loss: 0.8233, Validation Accuracy: 76.20%
Early stopping at epoch 7 out of 15
Saved best model with validation loss: 0.1309
<All keys matched successfully>
```

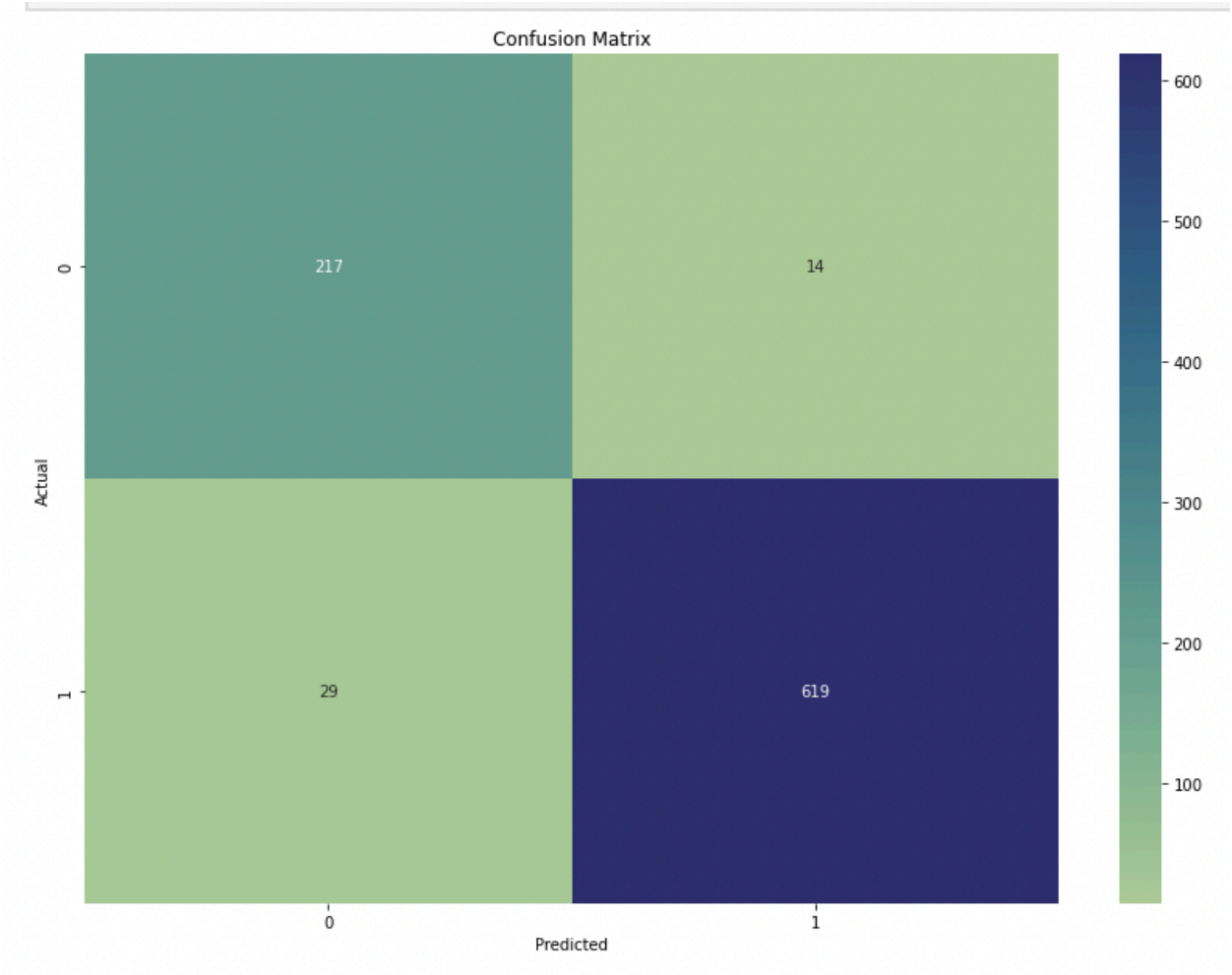
The below are the outputs for the evaluation metrics:
Test Accuracy: 95.11%
Final Precision: 0.9527
Final Recall: 0.9511
Final F1 Score: 0.9516

Visualization:

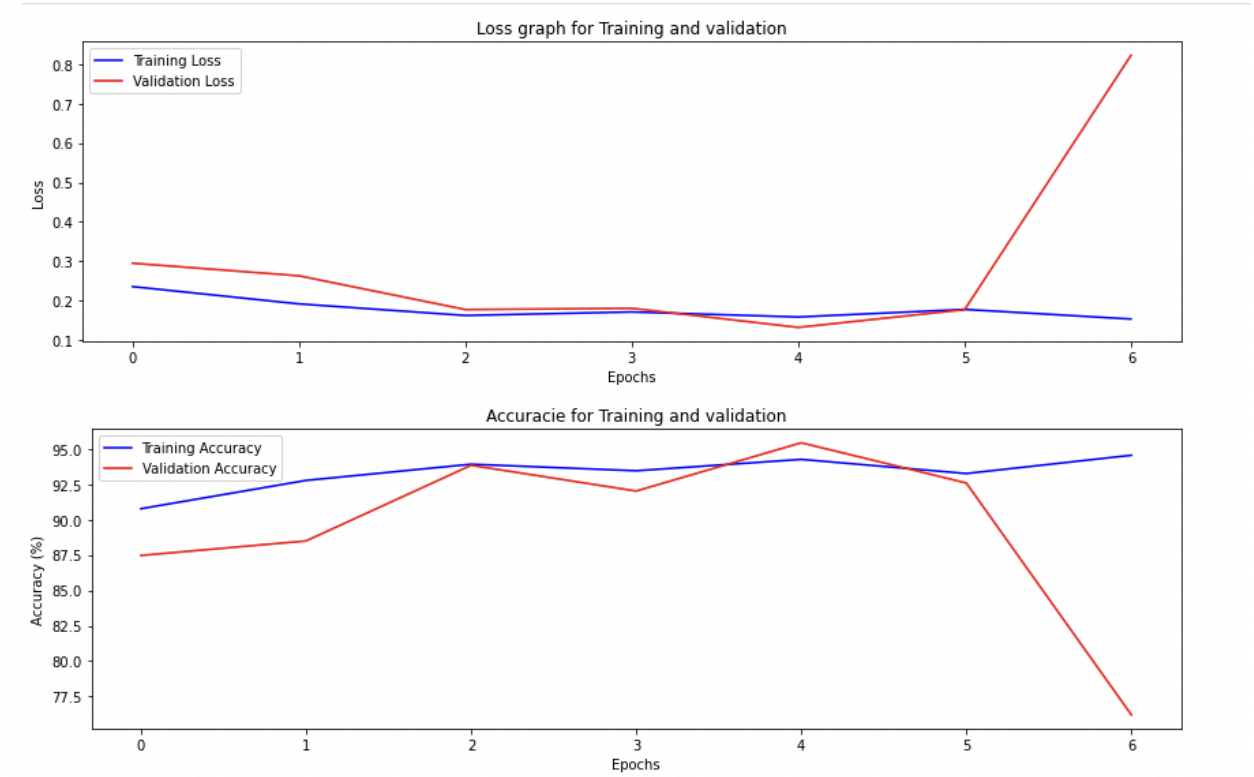
1) **Receiver Operating Characteristic:**



2) **Confusion Matrix:**



3) **Loss and Accuracy Plotting:**



Model 4: RESNET34

ResNet is a deep learning model that is used to make training simpler by using "skip connections" to bypass one or more layers. It eases the vanishing gradient problem. Due to this, the creation of very deep networks with improved accuracy is enabled.

Architecture:

- The fundamental building block of ResNet34 is the residual block. Each residual block includes two convolutional layers, along with a skip connection that directly adds the block's input to its output. This skip connection helps maintain gradient flow and enables the network to learn identity mappings, enhancing the training of deep networks.
- It consists of 4 layers of residual blocks where layer 1 has 3 blocks, layer 2 has 4 blocks, layer 3 has 6 blocks and layer 4 has 3 blocks.
- In the next part, it has a global average pooling which connects to a fully connected layer that is the output layer.

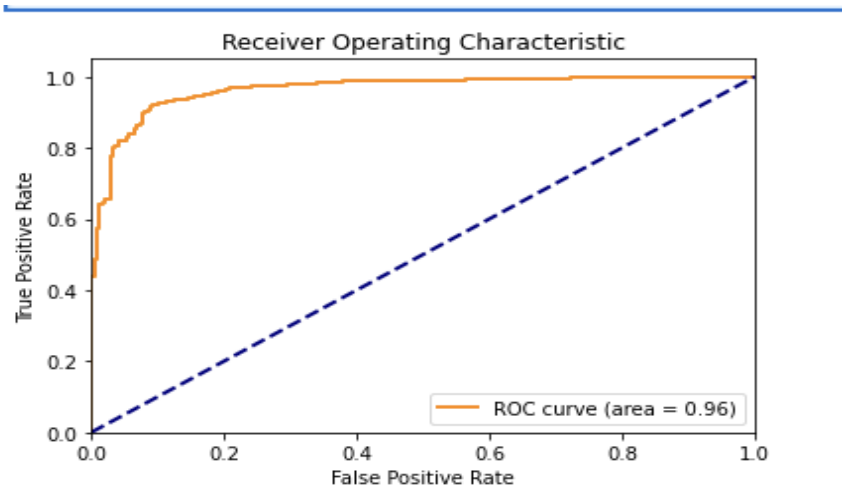
Training and Evaluation:

On training and evaluating the model,we get the following output for each epoch.

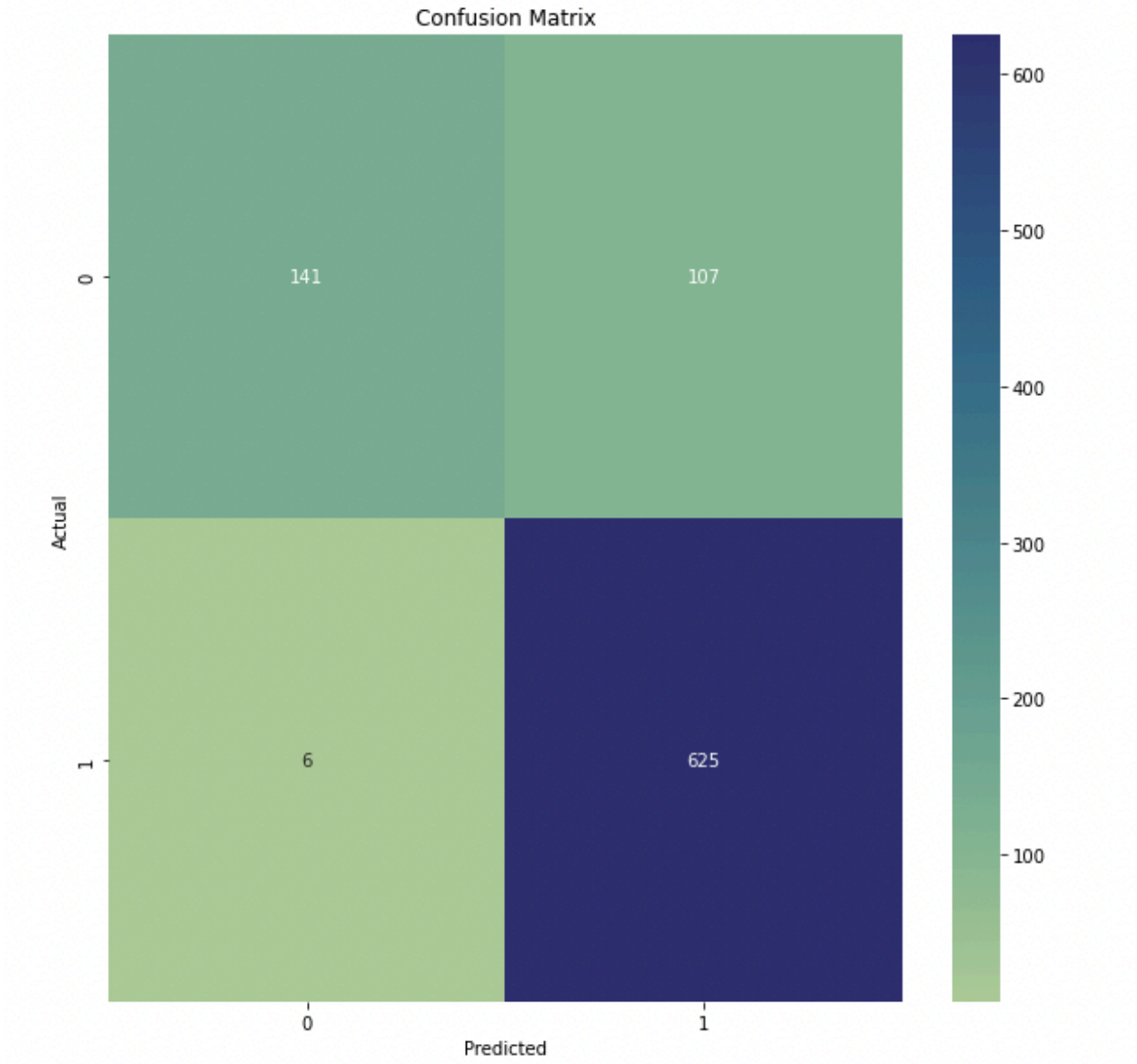
```
Epoch 1/10, Loss: 0.4482, Training Accuracy: 84.80%
Validation Loss: 0.2365, Validation Accuracy: 90.77%
Epoch 2/10, Loss: 0.3067, Training Accuracy: 86.56%
Validation Loss: 0.2782, Validation Accuracy: 88.72%
Epoch 3/10, Loss: 0.2660, Training Accuracy: 88.75%
Validation Loss: 0.4013, Validation Accuracy: 83.37%
Epoch 4/10, Loss: 0.2355, Training Accuracy: 90.34%
Validation Loss: 0.3143, Validation Accuracy: 83.37%
Epoch 5/10, Loss: 0.2525, Training Accuracy: 89.51%
Validation Loss: 0.2831, Validation Accuracy: 87.47%
Epoch 6/10, Loss: 0.2175, Training Accuracy: 91.29%
Validation Loss: 0.1822, Validation Accuracy: 93.39%
Epoch 7/10, Loss: 0.2115, Training Accuracy: 91.66%
Validation Loss: 0.2914, Validation Accuracy: 85.76%
Epoch 8/10, Loss: 0.2555, Training Accuracy: 89.78%
Validation Loss: 0.9227, Validation Accuracy: 69.59%
Epoch 9/10, Loss: 0.2470, Training Accuracy: 89.61%
Validation Loss: 0.1785, Validation Accuracy: 93.51%
Epoch 10/10, Loss: 0.2156, Training Accuracy: 91.61%
Validation Loss: 0.2259, Validation Accuracy: 90.77%
```

The following are the metrics for the model:

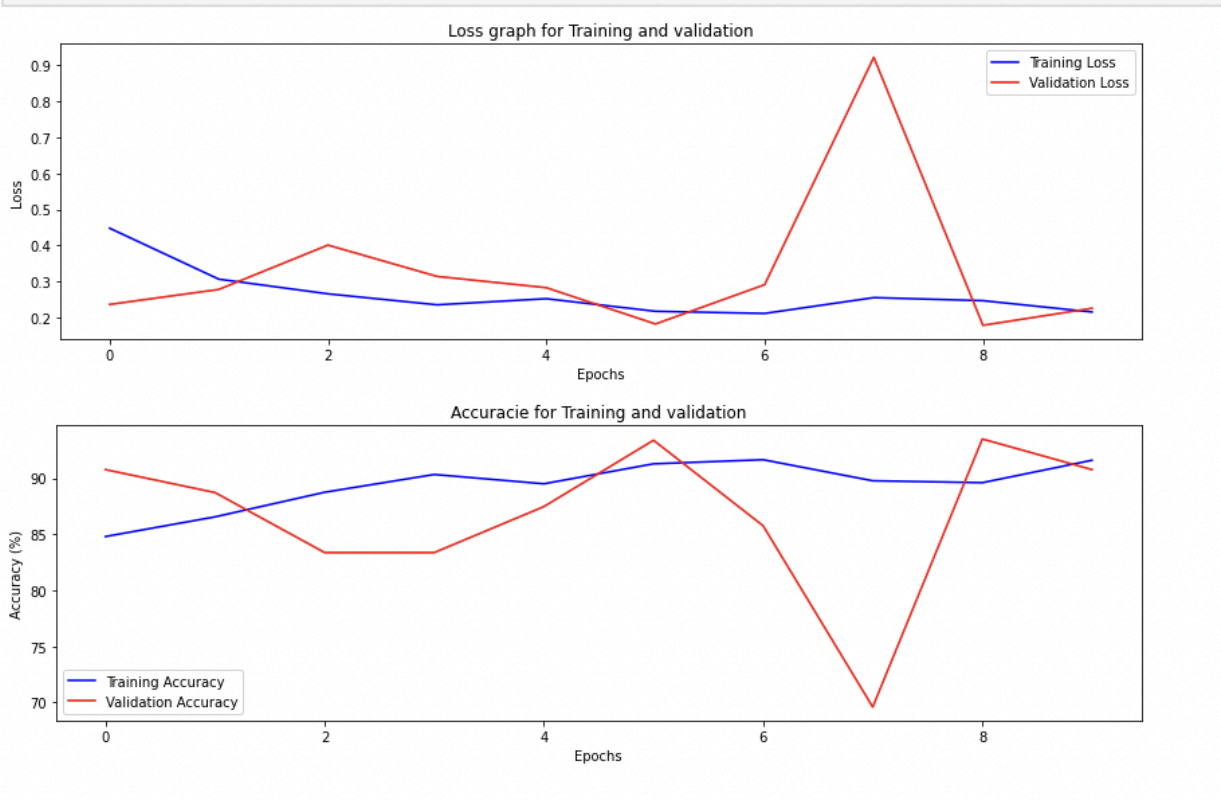
Test Accuracy: 87.14%
Final Precision: 0.8836
Final Recall: 0.8714
Final F1 Score: 0.8598
Visualizations:
1) Receiver Operating Characteristic



2) Confusion Matrix:



3) Loss and Accuracy Graphs:



Improved Resnet34 Model

In the improved Resnet34 model, we're using hyperparameter tuning. This is done by adding an L2 regularization. The other techniques that are added in the improved model are weight decay, early stopping and drop out layers. This helps to improve the overall performance of the model.

Training and Evaluation:

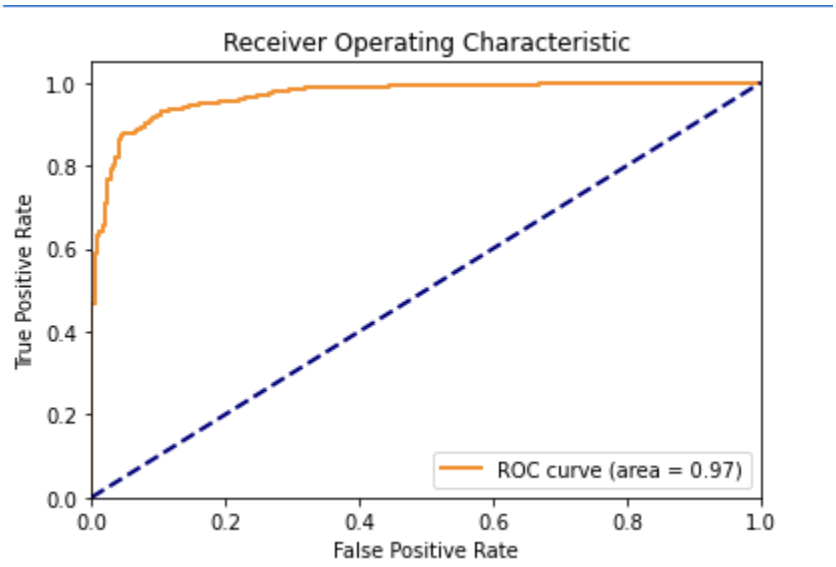
On training and evaluating the model, the following are the results:

```
Epoch 1/15, Loss: 0.4669, Training Accuracy: 83.95%
Validation Loss: 0.4471, Validation Accuracy: 77.11%
Epoch 2/15, Loss: 0.2785, Training Accuracy: 88.39%
Validation Loss: 0.3360, Validation Accuracy: 82.57%
Epoch 3/15, Loss: 0.2643, Training Accuracy: 89.34%
Validation Loss: 1.1354, Validation Accuracy: 75.17%
Epoch 4/15, Loss: 0.2590, Training Accuracy: 89.24%
Validation Loss: 0.1829, Validation Accuracy: 93.28%
Epoch 5/15, Loss: 0.2525, Training Accuracy: 89.66%
Validation Loss: 0.3186, Validation Accuracy: 85.42%
Epoch 6/15, Loss: 0.2349, Training Accuracy: 90.39%
Validation Loss: 0.3791, Validation Accuracy: 79.61%
Early stopping at epoch 6 out of 15
Saved best model with validation loss: 0.1829
[17…  <All keys matched successfully>
```

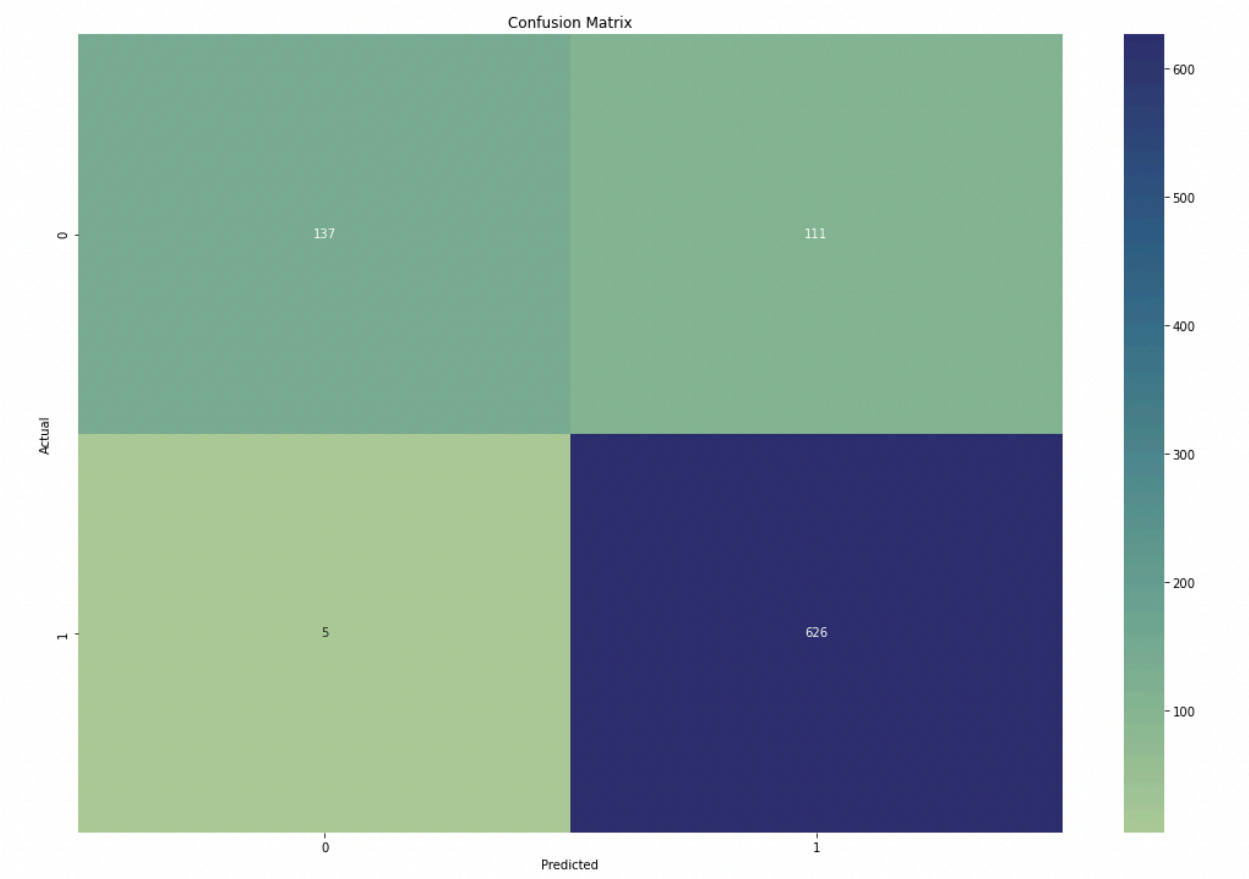
The overall metrics are:
Test Accuracy: 86.80%
Final Precision: 0.8819
Final Recall: 0.8680
Final F1 Score: 0.8552

Visualizations:

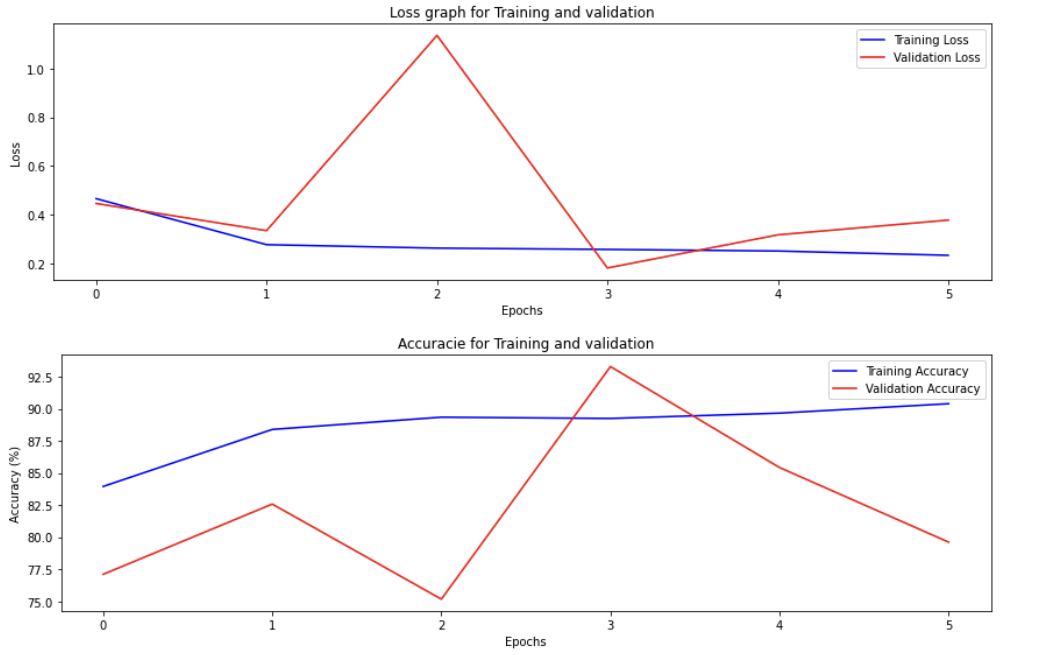
1) **Receiver Operating Characteristic Curve:**



2) **Confusion matrix:**

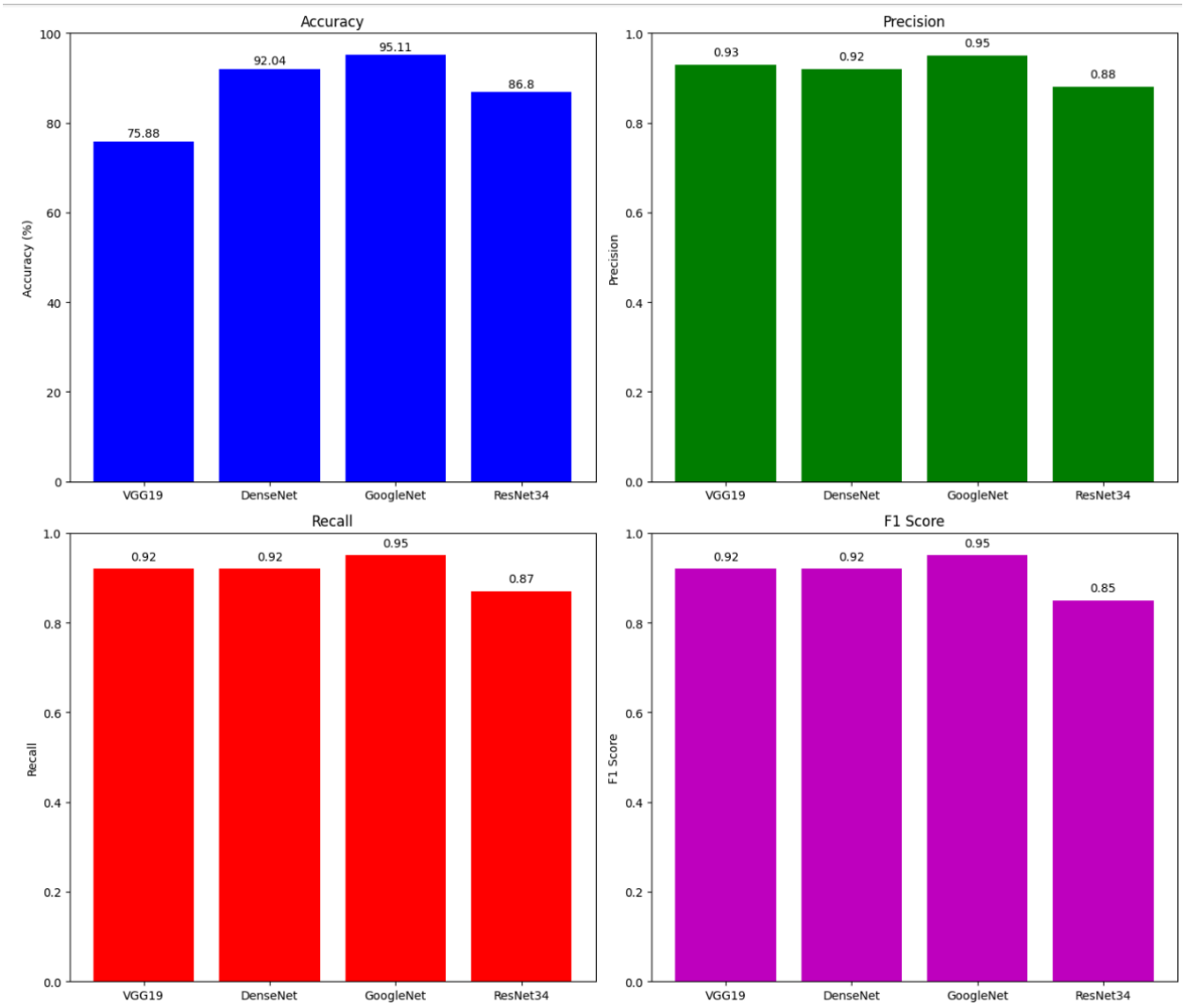


3) Accuracy and Loss:



Evaluation of the models:

	VGG19	DenseNet	GoogleNet	ResNet34
Accuracy	75.88%	92.04%	95.11%	86.80%
Precision	0.93	0.92	0.95	0.88
Recall	0.92	0.92	0.95	0.87
F1 score	0.92	0.92	0.95	0.85

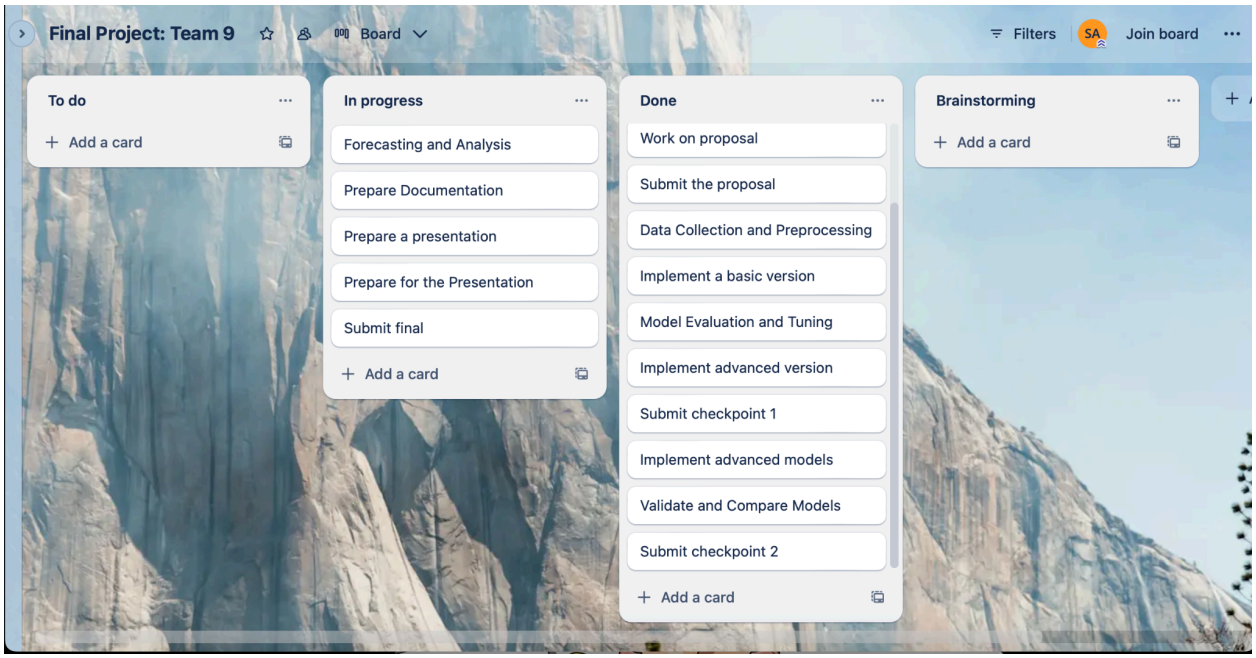


Based on the above performance metrics It can be seen that the pre trained model GoogleNet performed the best while VGG19 performed the worst among the 4 models. The reason why googlenet could be more accurate compared to the other models is because of the use of inception modules which uses multiple convolution layers of different sizes in parallel. This helps in capturing various details. Densenet connects each layer to every other layer, enhancing gradient flow, promoting feature reuse, and reducing parameters. Similarly, resnet34 performs better because of efficient gradient flow by introducing residual connections. By eliminating the vanishing gradient problem and its deep architecture helps resnet34 achieve remarkable accuracy and better performance in various other metrics.

While the other models perform similarly in terms of accuracy, VGG19 has the least accuracy because of the large number of parameters, unlike resnet and densenet it lacks better gradient flow mechanisms and feature reuse. Further it has less efficient architecture because of huge number of parameters.

Hence, in conclusion it can be said that GoogleNet model (pre trained) performed the best of the lot with an accuracy of 95.11% followed by DenseNet with an accuracy of 92.04%, ResNet34 with an accuracy of 86.80% and VGG19 is the least performing with 75.80%

Trello dashboard:



References:

1. Class Notes
2. https://github.com/dsgiitr/d2l-pytorch/blob/master/Ch09_Modern_Convolutional_Networks/Densely_Connected_Networks_(DenseNet).ipynb
3. https://pytorch.org/hub/pytorch_vision_densenet/
4. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8461289/
5. https://medium.com/@mukhriddinmalik/top-5-cnn-architectures-googlenet-resnet-densenet-alexnet-and-vggnet-to-build-your-computer-ca0c6f93512e
6. https://builtin.com/artificial-intelligence/resnet-architecture
7. https://amaarora.github.io/posts/2020-08-02-densenets.html
8. https://www.nature.com/articles/s41598-020-70479-z
9. https://towardsdatascience.com/understanding-and-visualizing-densenets-7f688092391a
10. https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8
11. https://medium.com/@siddheshb008/resnet-architecture-explained-47309ea9283d
12. https://medium.com/@AnasBrital98/vgg-16-and-vgg-19-cnn-architectures-d876f639cab7
13. https://python.plainenglish.io/densenet-densely-connected-convolutional-neural-networks-0fd219379138
14. https://medium.com/@siddheshb008/resnet-architecture-explained-47309ea9283d
15. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.RocCurveDisplay.html
16. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html
17. https://github.com/topics/googlenet?l=python&o=asc&s=stars
18. https://matplotlib.org/stable/plot_types
19. https://pytorch.org/tutorials/beginner/basics/data_tutorial.html
20. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
21. https://medium.com/codex/vggnet-complete-architecture-5c6fa801502b
22. https://pytorch.org/vision/stable/transforms.html
23. http://pytorch.org/vision/stable/models.html
24. https://pytorch.org/vision/main/models/densenet.html
25. https://trello.com/b/gqni3af1/final-project-team-9
26.

CSE 676: Deep Learning, Summer 2024

Instructor: Alina Vereshchaka

Final Report

Team Members
Hariharan Sriram (hsriram)
Swanith Ambadas (swanitha)
Manish Bikumalla (manishbi)

Introduction:

The aim of this project is to leverage deep learning models like CNN and apply them to chest X-rays image data to classify whether a person suffers from pneumonia or not. As part of the final submission, we tried to implement GAN and Autoencoders into our project and check how will these introductions effect our classification using the models that we implemented as part of checkpoint 1 (CNN) and checkpoint 2 (DenseNet and ResNet).

Steps covered in checkpoint 1:

We performed data processing and cleaning to the dataset. The data is split into training, testing and validation. Then applied CNN model and Improved CNN model to the dataset and trained the model to find the evaluation metrics such as Accuracy and Loss for training and testing and validate them over epochs.

Findings of checkpoint 1:

For the improved version of the CNN, the accuracy came out to be 92.84%. Based on other charts and performance metrics, the model seemed to have performed really well.

Steps covered in checkpoint 2:

As part of the checkpoint 2, we implemented 4 advanced models namely, VGG-19, ResNet-34, DenseNet and GoogleNet. We have implemented base versions of each of models and later made hyperparameter tuning, added dropout layers and weight decay along with early stopping to avoid overfitting. We compared the evaluation metrics of each and drew few conclusions.

Evaluation of the models:

	VGG19	DenseNet	GoogleNet	ResNet34
Accuracy	75.88%	92.04%	95.11%	86.80%
Precision	0.93	0.92	0.95	0.88

Recall	0.92	0.92	0.95	0.87
F1 score	0.92	0.92	0.95	0.85

Findings of checkpoint 2:

While the other models perform similarly in terms of accuracy, VGG19 has the least accuracy because of the large number of parameters, unlike resnet and densenet it lacks better gradient flow mechanisms and feature reuse. Further it has less efficient architecture because of huge number of parameters.

Hence, in conclusion it can be said that GoogleNet model (pre trained) performed the best of the lot with an accuracy of 95.11% followed by DenseNet with an accuracy of 92.04%, ResNet34 with an accuracy of 86.80% and VGG19 is the least performing with 75.80%.

Final Submission:

One of the main drawback/issues that we faced during our initial submissions is that every model took a lot of time in training (on an avg of 4000 to 6000 secs). This is because each image of ours is 224*224 pixels, therefore it is very difficult for any model to get hold of all the underlying features among all the pixels. This resulted in high training times. To mitigate this, we came up with an idea of using low dimensional encoded images generated from an autoencoder on the same CNN that we developed for checkpoint 1.

Autoencoder:

Autoencoders are a type of artificial neural network used to learn efficient representations of data, typically for the purpose of dimensionality reduction, feature extraction, or data generation. They are unsupervised learning models, meaning they do not require labeled data to train.

Structure of Autoencoders

Autoencoders consist of two main components:

1. **Encoder**: This part of the network compresses the input data into a lower-dimensional representation (also known as the latent space or code). The encoder typically consists of a series of layers that reduce the dimensionality of the input data.
2. **Decoder**: This part of the network reconstructs the original input data from the lower-dimensional representation produced by the encoder. The decoder typically consists of a series of layers that increase the dimensionality of the latent space back to the original input size.

Input → Encoder → Latent Space → Decoder → Output (Reconstructed Input)

Advantages of Autoencoders

1. **Dimensionality Reduction**:
    - **Automatic Feature Extraction**: Autoencoders can learn to extract meaningful features from raw data without the need for manual feature engineering.
    - **Data Compression**: They can compress data into a lower-dimensional representation, which can be useful for data storage and transmission.
2. **Noise Reduction**:
    - **Denoising**: Autoencoders can be trained to remove noise from data, making them useful for image denoising and other noise reduction tasks.
3. **Anomaly Detection**:
    - **Unsupervised Learning**: They can be used to detect anomalies in data by identifying inputs that significantly deviate from the learned representation.

Disadvantages of Autoencoders

1. **Overfitting**:
    - **Data Dependency**: Autoencoders can easily be overfit to the training data, especially when the network is too complex or the training data is limited.
2. **Lack of Interpretability**:
    - **Latent Space Complexity**: The learned representations in the latent space can be difficult to interpret and may not have a clear or intuitive meaning.
3. **Training Challenges**:
    - **Training Instability**: Training autoencoders can be challenging due to issues like vanishing gradients and the need for careful tuning of hyperparameters.

Autoencoders as a pre-processing step:

Using encoded low-dimensional images for model training significantly enhances efficiency by reducing computational complexity and memory usage. This compression is achieved through autoencoders, which extract essential features and discard irrelevant information, leading to faster computations and shorter training times. The smaller data size also helps prevent overfitting and improves generalization to new data by removing noise and focusing on key patterns. Furthermore, the simplified models require less hyperparameter tuning, contributing to easier maintenance and faster training cycles. Pre-trained encoders enable efficient transfer learning, allowing models to adapt to new tasks or domains with minimal retraining. Overall, this approach streamlines the training process while maintaining or even enhancing model performance.

Autoencoder Architecture:

- **Conv2d(1, 32, kernel_size=3, stride=2, padding=1)**: Input channels = 1, output channels = 32, kernel size = 3, stride = 2, padding = 1.
- **BatchNorm2d(32)**: Batch normalization over 32 feature maps.
- **ReLU()**: Activation function.
- **Conv2d(32, 64, kernel_size=3, stride=2, padding=1)**: Input channels = 32, output channels = 64.
- **BatchNorm2d(64)**: Batch normalization over 64 feature maps.
- **ReLU()**: Activation function.
- **Conv2d(64, 128, kernel_size=3, stride=2, padding=1)**: Input channels = 64, output channels = 128.
- **BatchNorm2d(128)**: Batch normalization over 128 feature maps.
- **ReLU()**: Activation function.
- **Conv2d(128, 256, kernel_size=3, stride=2, padding=1)**: Input channels = 128, output channels = 256.
- **BatchNorm2d(256)**: Batch normalization over 256 feature maps.
- **ReLU()**: Activation function.

Criterion (Loss Function)

1. **Mean Squared Error (MSE) Loss**: The criterion used in this autoencoder is `nn.MSELoss()`, which calculates the mean squared error between the reconstructed output and the original input. This loss function is appropriate for regression tasks and ensures that the autoencoder learns to accurately reconstruct the input data.

Optimizer

1. **Adam Optimizer**: The optimizer used is `optim.Adam()`, which is an adaptive learning rate optimization algorithm designed to handle sparse gradients on noisy problems. It combines the advantages of two other extensions of stochastic gradient descent, AdaGrad and RMSProp, making it suitable for complex neural networks.

Steps Involved :

Our image size is 224*224 pixels which have only 1 Gray-scale channel. Training a model using such large images is a time taking process. So, we applied an encoder which takes 224*224 gray-scale image and produces 14*14 encoded images with 256 layers.

During encoding of images using autoencoder the input image is passed through several convolutional layers. Each convolutional layer applies a set of filters to the input, capturing different features of the image such as edges, textures, and patterns.

Using these encoded images reduced the time of training while keeping the accuracy and precision intact.

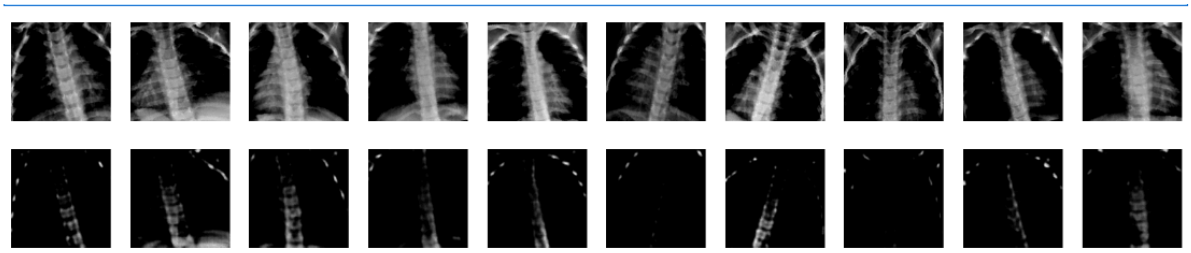
Here is the reconstruction loss while training and validation of the images over 10 epochs.

```
Epoch [1/10], Train Loss: 0.5710390895402351, Validation Loss: 0.5153604196874719
Epoch [2/10], Train Loss: 0.49194409206610956, Validation Loss: 0.5003079050465634
Epoch [3/10], Train Loss: 0.48513503403079755, Validation Loss: 0.4971582403308467
Epoch [4/10], Train Loss: 0.4832642682555581, Validation Loss: 0.49471581452771235
Epoch [5/10], Train Loss: 0.4815977578665934, Validation Loss: 0.4957249133210433
Epoch [6/10], Train Loss: 0.4811216943523511, Validation Loss: 0.4940932095050812
Epoch [7/10], Train Loss: 0.48102144866573565, Validation Loss: 0.49289965002160324
Epoch [8/10], Train Loss: 0.48016146088944, Validation Loss: 0.49485525488853455
Epoch [9/10], Train Loss: 0.48052999316429607, Validation Loss: 0.4924440195685939
Epoch [10/10], Train Loss: 0.4803122977415721, Validation Loss: 0.49304669781735067
```

Final output of autoencoder (after both encoding and decoding steps producing same size images as input 224*224)

Row 1 : Original images

Row 2 : Autoencoder generated images



We sent the encoded images into our Adv_CNN that we developed as part of our checkpoint 1.

Adjusted the the CNN to accept 14*14 images with 256 layers each and then classify these images as ‘normal’ or ‘pneumonia’ class.

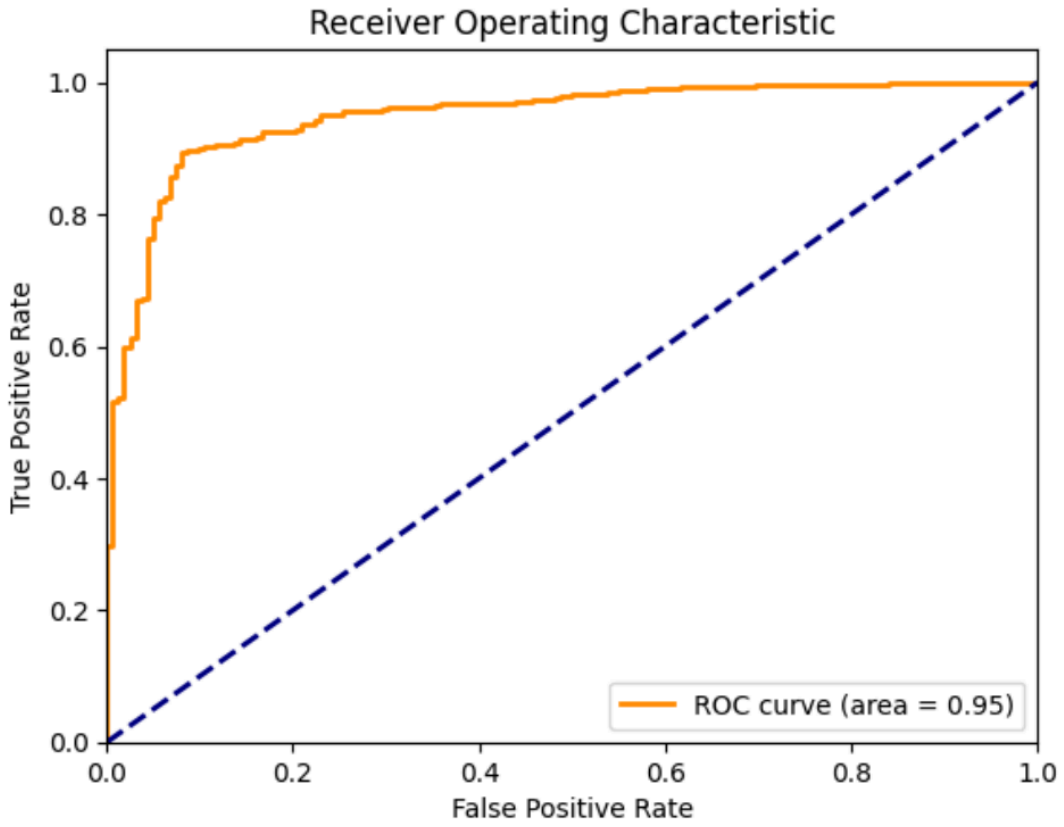
From this CNN we had a maximum validation accuracy of 90.77% and execution was drastically reduced to just 17.52 seconds.

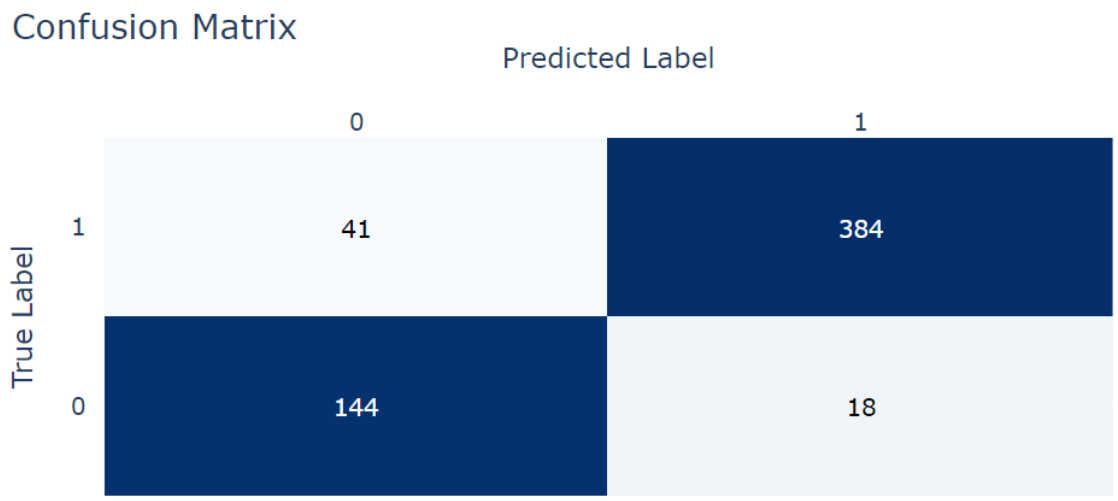
```
Epoch 1: Train Loss: 0.4633, Train Acc: 76.71%, Val Loss: 0.3759, Val Acc: 86.50%
Epoch 2: Train Loss: 0.3505, Train Acc: 85.67%, Val Loss: 0.3666, Val Acc: 83.59%
Epoch 3: Train Loss: 0.3259, Train Acc: 86.87%, Val Loss: 0.2969, Val Acc: 87.69%
Epoch 4: Train Loss: 0.3158, Train Acc: 87.60%, Val Loss: 0.3361, Val Acc: 89.91%
Epoch 5: Train Loss: 0.2807, Train Acc: 89.24%, Val Loss: 0.3063, Val Acc: 90.26%
Epoch 6: Train Loss: 0.2909, Train Acc: 88.88%, Val Loss: 0.3928, Val Acc: 84.96%
Epoch 7: Train Loss: 0.2796, Train Acc: 89.28%, Val Loss: 0.3776, Val Acc: 87.69%
Epoch 8: Train Loss: 0.2893, Train Acc: 89.01%, Val Loss: 0.4091, Val Acc: 88.55%
Epoch 9: Train Loss: 0.2814, Train Acc: 89.37%, Val Loss: 0.3608, Val Acc: 88.38%
Epoch 10: Train Loss: 0.2646, Train Acc: 90.29%, Val Loss: 0.2878, Val Acc: 90.77%
Total training time: 17.52 seconds
```

Here are the metrics on test data

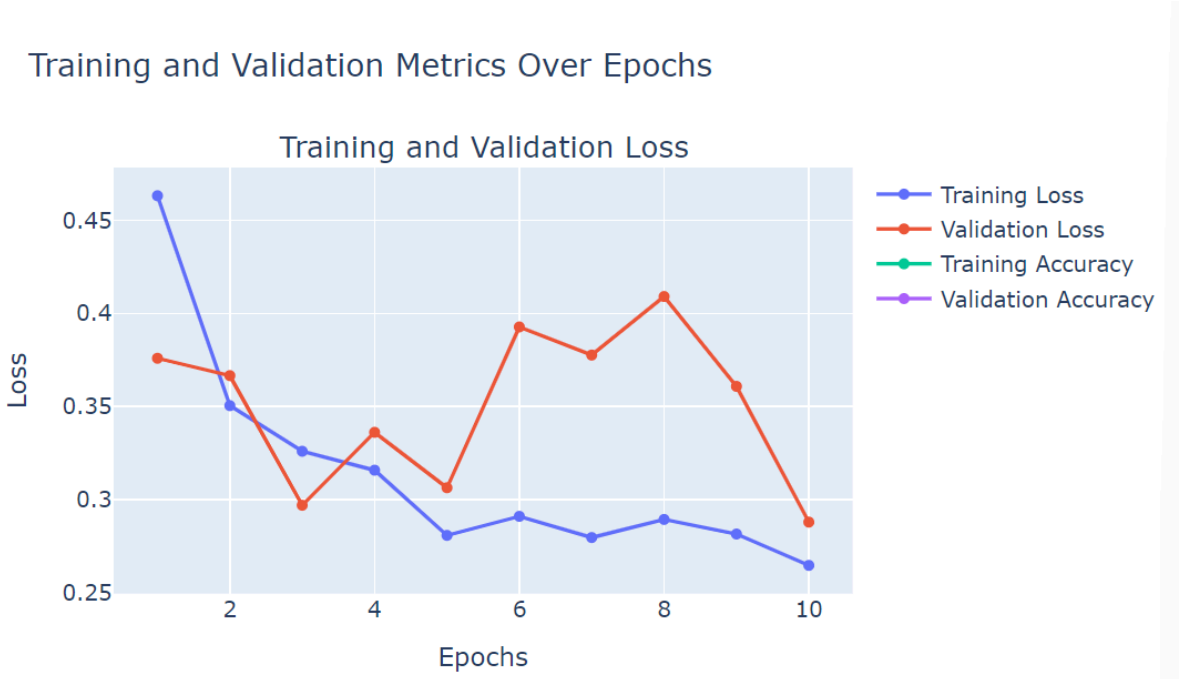
```
Test Loss: 0.2966, Test Accuracy: 89.95%
Precision: 0.9064, Recall: 0.8995, F1 Score: 0.9014
```

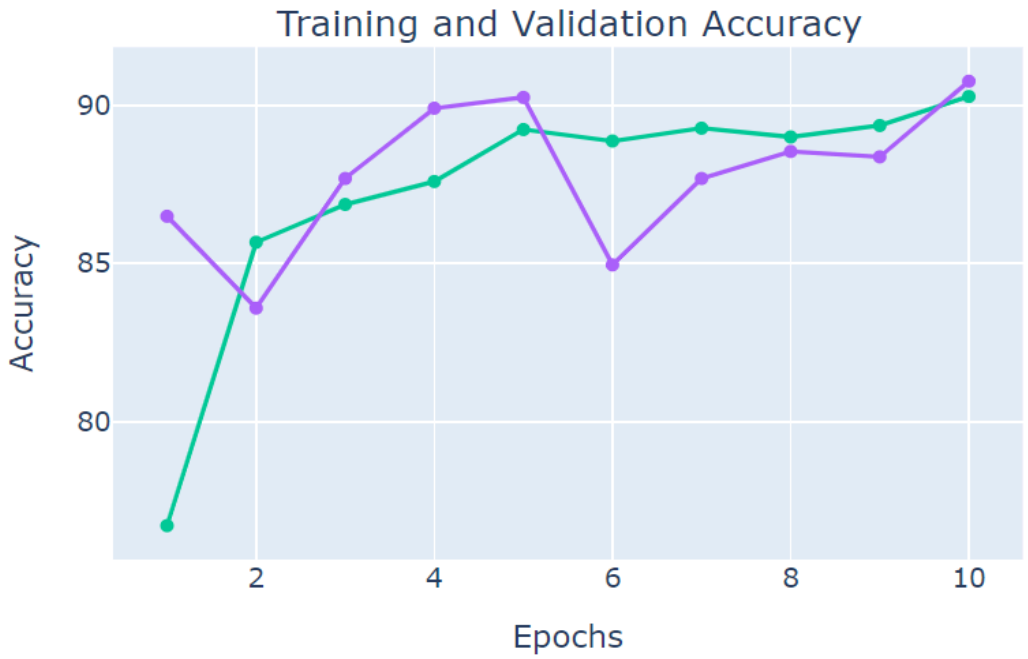
ROC and Confusion matrix :





Training and Validation losses and accuracies over the course of 10 epochs:





Conclusion

Initially, we employed an improved CNN model, achieving a notable accuracy of 92.84%. However, high training times due to the large image size of 224x224 pixels presented a significant challenge. To address this, we introduced autoencoders to compress the images into a lower-dimensional representation, specifically 14x14 with 256 channels.

The autoencoder effectively reduced the dimensionality, preserving essential features and reducing computational load. Using these encoded images for training significantly lowered training times while maintaining high performance. The adjusted CNN model, utilizing the encoded images, achieved a validation accuracy of 90.77% with a training time reduced to just 17.52 seconds.

Our evaluation metrics on test data confirmed the robustness of this approach. The ROC curve and confusion matrix demonstrated high precision and recall, indicating reliable classification. Additionally, the training and validation loss and accuracy graphs over ten epochs showed stable convergence, further validating our method.

In conclusion, leveraging autoencoders for dimensionality reduction proved to be an effective strategy to enhance training efficiency without compromising accuracy. This approach can be highly beneficial in similar tasks where large image sizes pose a computational challenge.

GAN (Generative adversarial Networks):

What are they?

GANs are a class of neural networks introduced by Ian Goodfellow and his colleagues in 2014. Its architecture consists of 2 neural networks – generator and discriminator. Generator part generates new synthetic images and discriminator part evaluates the authenticity of the data by differentiating between the real data and fake data generated. This process overtime leads to produce increasingly accurate data over time.

The Generator's primary function is to transform random noise into convincing, lifelike data. This process involves a sequence of transposed convolutional layers, which are sometimes referred to as deconvolutional layers. These specialized layers work to expand the initial random noise input, gradually increasing its size and complexity. The result is a larger, more intricate image that closely mimics real-world data.

The Discriminator's purpose is to differentiate between genuine images from the dataset and fake images produced by the Generator. It employs a sequence of convolutional layers to reduce the dimensionality of the input image, ultimately making a binary classification to determine whether the image is real or fake.

Major issues and the need for GANs when classifying this dataset-

- Limited data: Chest Xray dataset is limited in size because of various reasons like privacy, lack of personnel for collecting the data and the difficulty of obtaining labelled data. By using GANs the data can be expanded by generating artificial images
- Imbalance: This dataset suffers from class imbalance like some class being underrepresented. Hence, GAN based data generation can help circumvent this problem
- Improved model performance is possible especially when underrepresented classes have an increased number of imagery
- Problem of overfitting may occur when the size of the dataset is too small and the model struggles to generalize and accurately classify when presented with new data. Hence, by using GANs expansion of the original dataset is possible leading to reduction of overfitting problem.

Advantages:

- Leads to production of high-resolution data
- Does not require labelled data. Hence, suitable for unsupervised learning
- Can generate synthetic data as well as augment the existing data
- Highly versatile and can be used in various types of applications

Disadvantages:

- Difficulty in training – various problems like vanishing gradients, collapse of the model etc
- High computational cost – significant memory consumption
- GANs can overfit the training data, producing synthetic data that is too like the training data and thereby lacking in diversity

- Difficulty in evaluation of the generated images
- Highly sensitive to hyperparameters
- Ethical and privacy concerns

However, when compared to the variational auto encoders, they produce higher quality and more photorealistic images. Despite their limitations, GANs remain a powerful and widely used tool in machine learning. Their benefits far outweigh the limitations especially for the chest xray datasets where there is a paucity of data. Further, with focused research in the scientific community the shortcomings are likely to be addressed pushing the boundaries of data synthesis and manipulation.

GAN Model Architecture:
Generator:
The model takes a 100-dimensional noise vector as input. It uses transposed convolutions to upsample the input. It was structured in such a way that it goes from small and deep (100x1x1) to large and shallow (1x64x64). Each layer except the last layer has batch normalization and ReLU activation function. The final layer uses Tanh activation function. The output is a 64x64 grayscale image.

Discriminator:
Takes the output generated by the generator i.e. 64x64 image as input. Unlike the generator it uses regular convolutions to downsample the input. Unlike the generator, its structure goes from large and shallow (1x64x64) to small and deep. Each layer except the first and last have a batch normalization and LeakyReLU activation function. The final layer flattens the output and then passes the data through Sigmoid function.

The model was run twice once for NORMAL images and the second time for PNEUMONIA images. At the end of each epoch 64 images are generated and since there are 20 epochs, a total of 1280 images are generated each for NORMAL and PNEUMONIA data. Thereby a total of 2560 images are generated. This enhanced the size of the dataset from 5856 images to 8416 images.

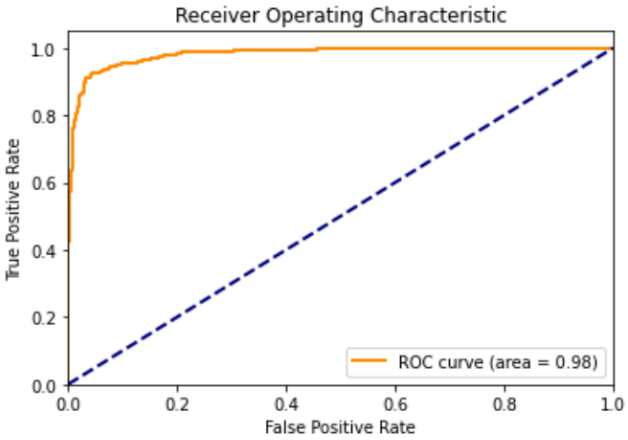
Application:
These synthetic data are combined with the original data and the previously created resnet-34 and densenet models are run again to classify the chest xray datasets.

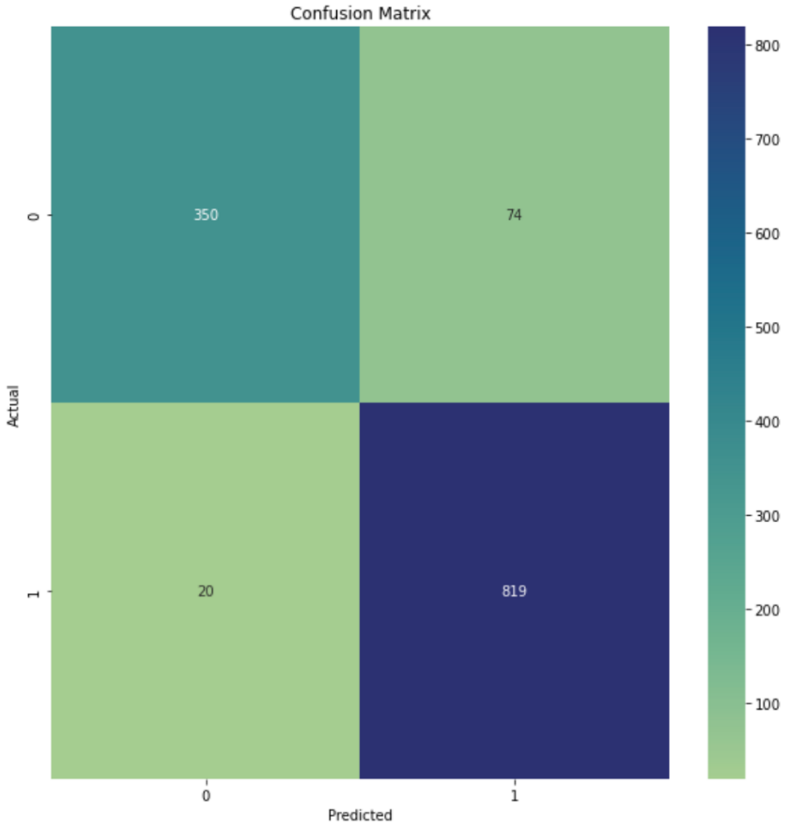
Split of the dataset

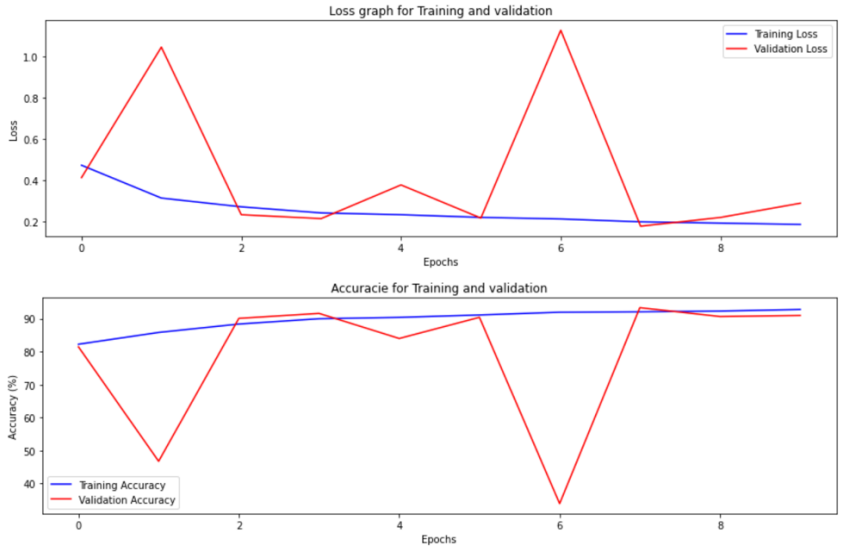
```
Total samples: 8416
Train samples: 5891
Validation samples: 1262
Test samples: 1263
```

Application on RESNET-34:
Below are the evaluation metrics as well as variation visualisations for the base model

Test Accuracy: 92.56%
Final Precision: 0.9268
Final Recall: 0.9256
Final F1 Score: 0.9242

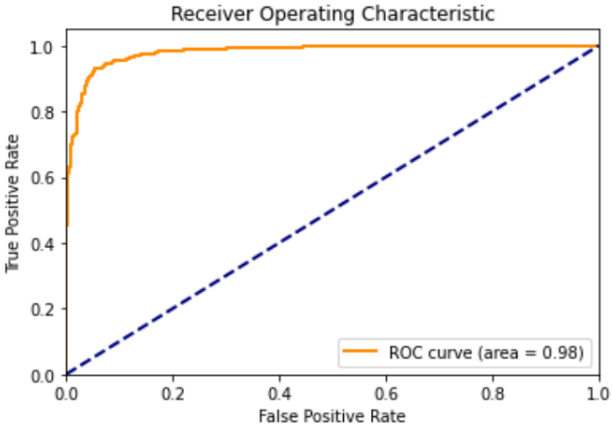


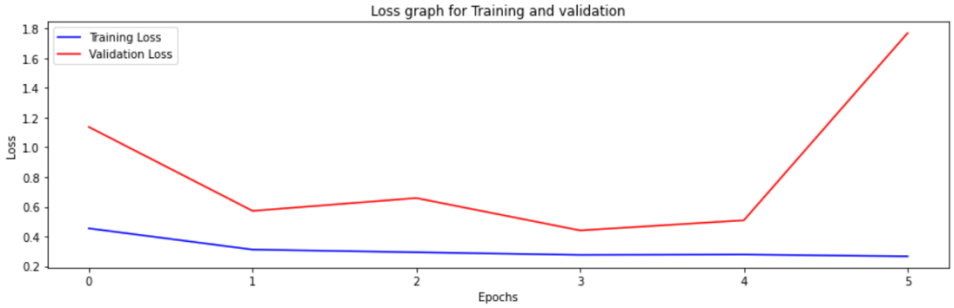


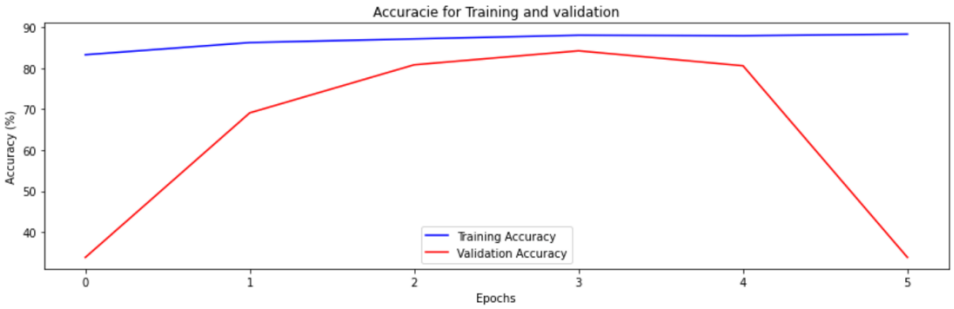


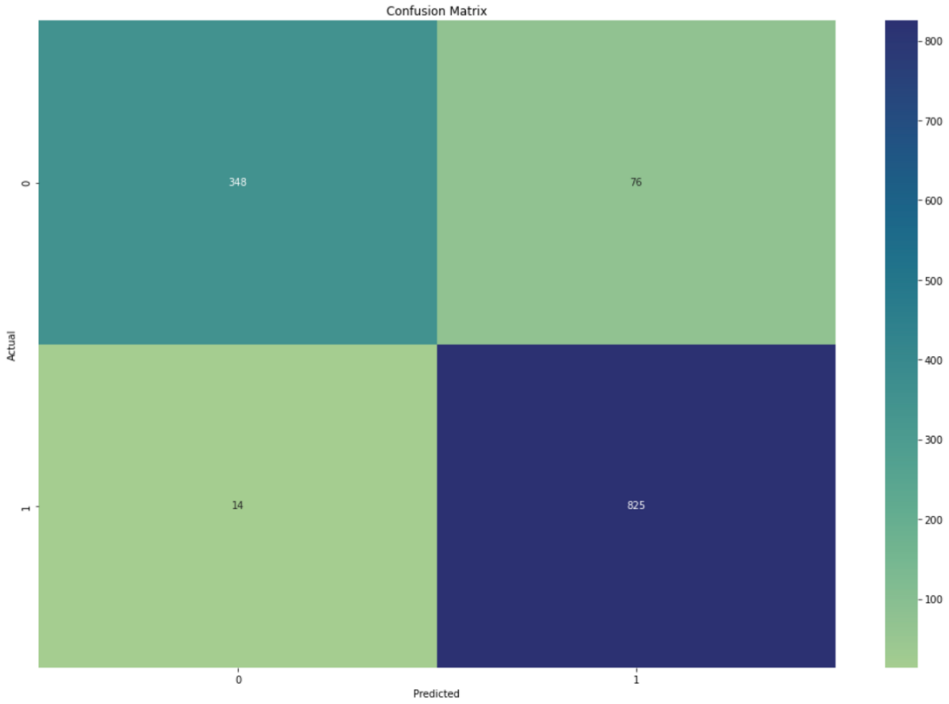
The model was further improved by adding L2 regularisation, early stopping and dropout layers. Further below are the evaluation metrics and visualisations for the improved model.

```
Test Accuracy: 92.87%
Final Precision: 0.9310
Final Recall: 0.9287
Final F1 Score: 0.9272
```





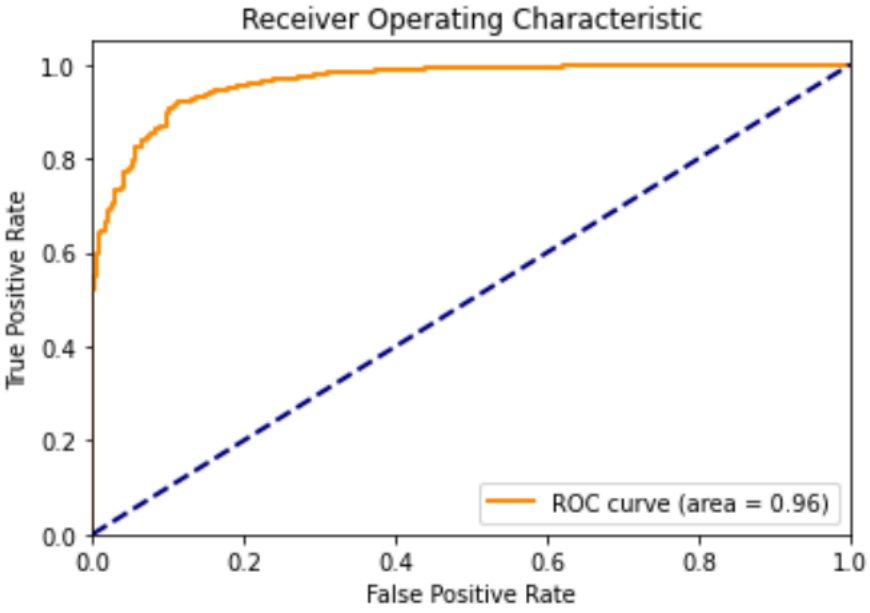


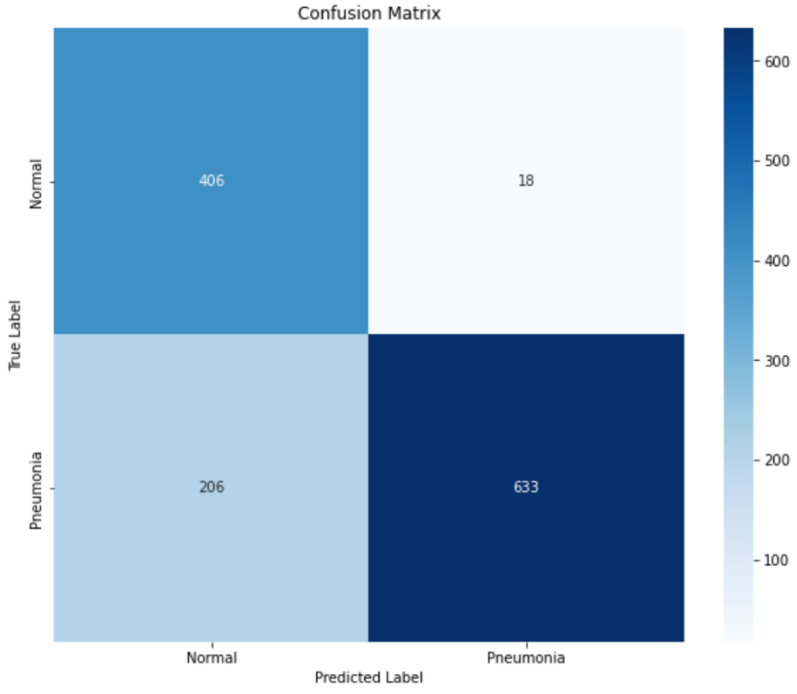


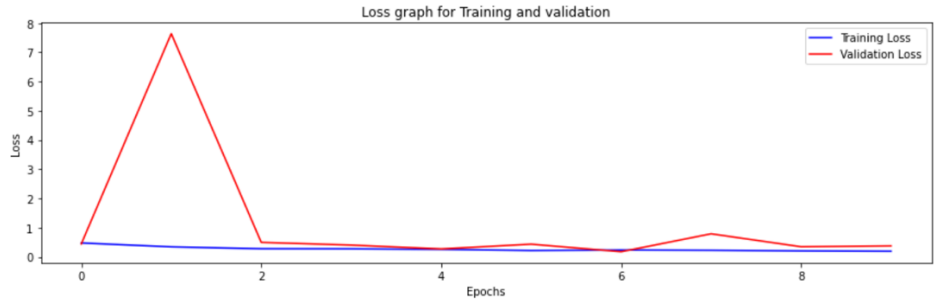
A slight increase in the test accuracy as well as other metrics is observed compared to the base model.

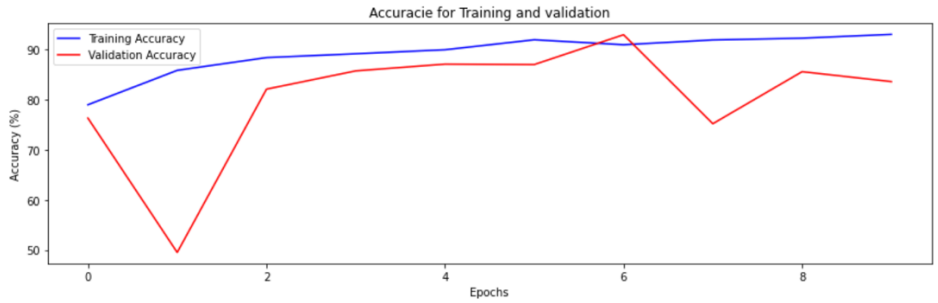
Application on DenseNet model:
Similarly the analysis was performed on the DenseNet model and below are the evaluation metrics for base model

```
Test Accuracy: 82.26%
Final Precision: 0.8686
Final Recall: 0.8226
Final F1 Score: 0.8275
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.96)
0.0
0.2
0.4
0.6
0.8
1.0

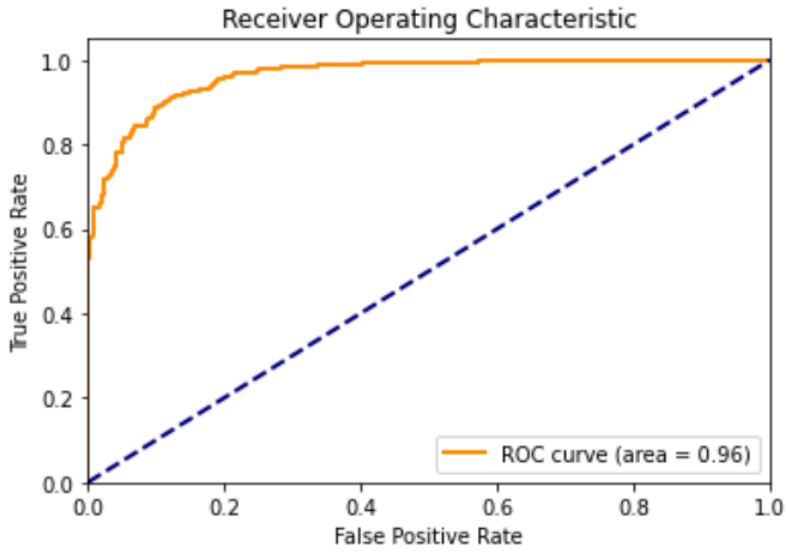

Confusion Matrix
True Label
Predicted Label
Normal
Pneumonia
406
18
206
633
600
500
400
300
200
100

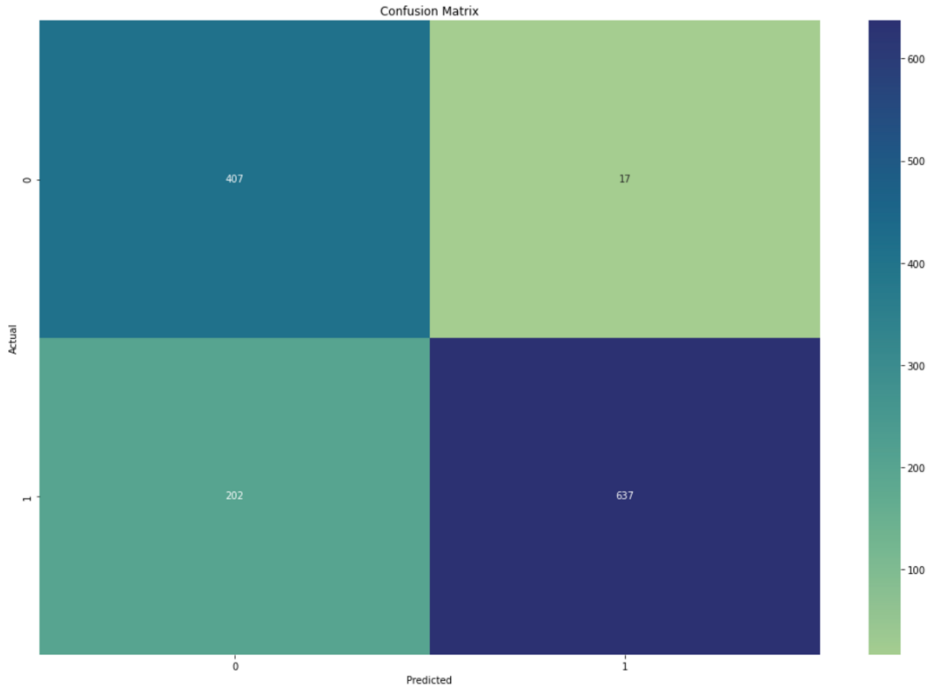


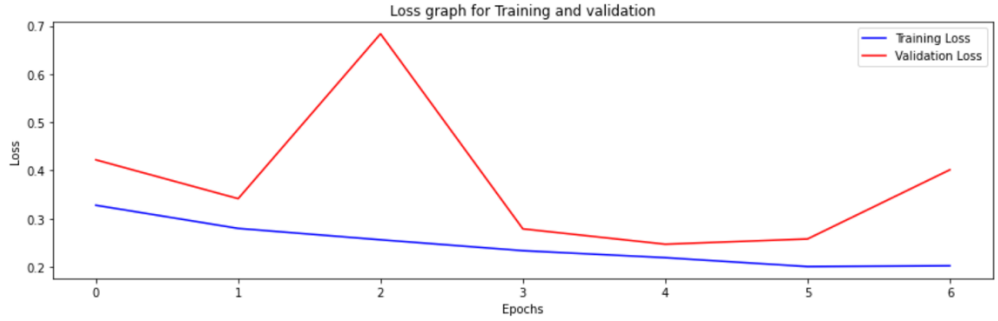


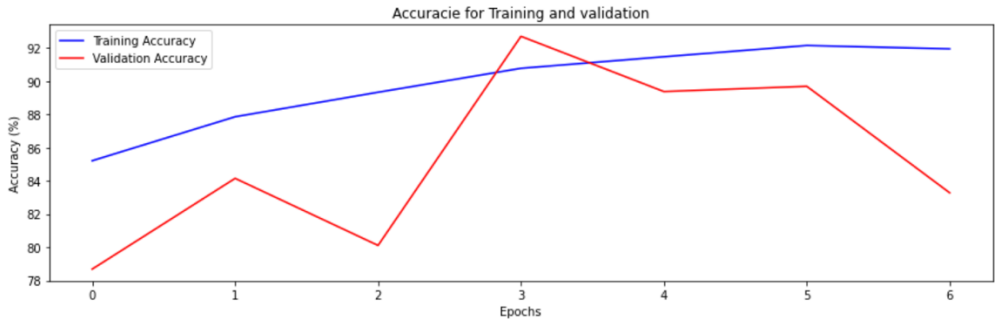
Upon addition of dropout, early stopping and L2 regularisation, below are the evaluation metrics

```
Test Accuracy: 82.66%
Final Precision: 0.8714
Final Recall: 0.8266
Final F1 Score: 0.8314
```









Comparative analysis of GAN based images and original dataset

	RESNET-34	DENSENET	RESNET-34 - GAN	DENSENET - GAN
Accuracy	86.80%	92.04%	92.87%	82.66%

Precision	0.88	0.92	0.93	0.87
Recall	0.87	0.92	0.93	0.83
F1 score	0.85	0.92	0.93	0.83

Analysis and inferences:
Based on the above comparision it can be observed that RESNET performs better when GAN generated images are introduced into the dataset while densenet performs worse when the dataset in introduced. It further can be observed that GANs might produce images with certain biases or patters that align better with Resnet's architecture. Resnet's skip connections might be better at handling the characteristics of GAN- generated images and they might be more resilient to any artifacts or slight inconsistencies in the GAN generated images. In the case of densenet, the generated images may lack some subtle features which are present in real images which Densenet might rely on heavily. The model lacks some fine grained details that densenet typically exploits. Also, dense connections of this model might be more sensitive to any noise or artifacts in the GAN generated images. This problem can be further analysed by using different architectures, identification of patters/artifacts if any, experimentation with different ratios of real vs generated images etc.

References:

- https://ar5iv.labs.arxiv.org/html/2107.02970
- Deep Learning (2017 edition)– Ian Goodfellow, Yoshua Bengio, Aaron Courville
- https://www.wallstreetmojo.com/generative-adversarial-network/
- https://www.clickworker.com/ai-glossary/generative-adversarial-networks/
- https://ai.stackexchange.com/questions/25601/what-are-the-fundamental-differences-between-vae-and-gan-for-image-generation
- https://resources.pcb.cadence.com/blog/2023-gan-advantages-and-disadvantages
- https://www.simplilearn.com/tutorials/deep-learning-tutorial/generative-adversarial-networks-gans
- https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b
- https://d2l.ai/chapter_computer-vision/transposed-conv.html
- https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11
- https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc
- https://medium.com/analytics-vidhya/downsampling-and-upsampling-of-images-demystifying-the-theory-4ca7e21db24a
- https://dsp.stackexchange.com/questions/71685/upsampling-vs-downsampling-which-to-use-when#:~:text=Downsampling%20reduces%20dimensionality%20of%20the,the%20resolution%20of%20previous%20layer
- Class Notes

AlexNet:
Alexnet is a type of convolutional neural network model architecture which is used in deep learning models which is used for image classification. Its main significance comes from its use in computer vision. It can help us decode complex image classification. It has deep convolutional layers with ReLU activation functions followed by max pooling after every layer. It is then followed by fully connected layers that are used for classification.

Base Model:
Architecture:
TheAlexNet model has 5 convolutional layers which are followed by activation functions and max pooling after each layer.

The first convolution layer has 64 filters with a kernel size of 11x11 and stride as 4 and padding 2. It is followed by a ReLU activation layer and max pooling.
The second convolution layer has 192 filters with a kernel size of 5x5 and padding 2. It is followed by a ReLU activation layer and max pooling.
The third convolution layer has 384 filters with a kernel size of 3x3 and padding 1. It is followed by a ReLU activation layer and max pooling.
The forth convolution layer has 256 filters with a kernel size of 3x3 and padding 1. It is followed by a ReLU activation layer and max pooling.
The last convolution layer has 256 filters with a kernel size of 3x3 and padding 1. It is followed by a ReLU activation layer and max pooling.

It has 3 linear layers with a dropout layer added after every linear layer followed by ReLU activation function.

Padding is used to maintain the dimensions of the input image from one convolution layer to another. Without padding, the dimension size might reduce after every layer which results in loss of information.
Max pooling is used to reduce the dimensions of the input images. This is used to prevent over-fitting. It helps us extract important features from the input data.

Loss Function:
The loss function used here is nn.CrossEntropyLoss(). It is mainly used for classification of the model. It measures the difference between the predicted labels and actual labels.

Optimiser:
Here, we're using the optim.Adam(model.parameters(), lr=0.001)with a learning rate of 0.001. The learning rate is used to control the size when the optimiser uses it to reduce

the loss function. The optimiser updates the parameters to improve the performance of the model by reducing the loss.

Structure of AlexNet:

```
BaseAlexNet(
  (features): Sequential(
    (0): Conv2d(1, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=2, bias=True)
  )
)
```

Training and Evaluation:

We’re training and evaluating the model by finding the Training loss, training accuracy, validation loss and validation accuracy over epochs.

```
Epoch 1/10, Train Loss: 0.5833, Train Acc: 0.7431, Val Loss: 0.3585, Val Acc: 0.8030
Epoch 2/10, Train Loss: 0.3540, Train Acc: 0.8273, Val Loss: 0.2984, Val Acc: 0.8497
Epoch 3/10, Train Loss: 0.3040, Train Acc: 0.8707, Val Loss: 0.2694, Val Acc: 0.8793
Epoch 4/10, Train Loss: 0.2776, Train Acc: 0.8880, Val Loss: 0.2266, Val Acc: 0.9112
Epoch 5/10, Train Loss: 0.2384, Train Acc: 0.9031, Val Loss: 0.2285, Val Acc: 0.9077
Epoch 6/10, Train Loss: 0.2205, Train Acc: 0.9134, Val Loss: 0.2344, Val Acc: 0.9112
Epoch 7/10, Train Loss: 0.2054, Train Acc: 0.9214, Val Loss: 0.1763, Val Acc: 0.9294
Epoch 8/10, Train Loss: 0.1912, Train Acc: 0.9300, Val Loss: 0.1863, Val Acc: 0.9237
Epoch 9/10, Train Loss: 0.1803, Train Acc: 0.9290, Val Loss: 0.1936, Val Acc: 0.9191
Epoch 10/10, Train Loss: 0.2113, Train Acc: 0.9171, Val Loss: 0.1833, Val Acc: 0.9282
Training complete in 13m 16s
```

Metrics:

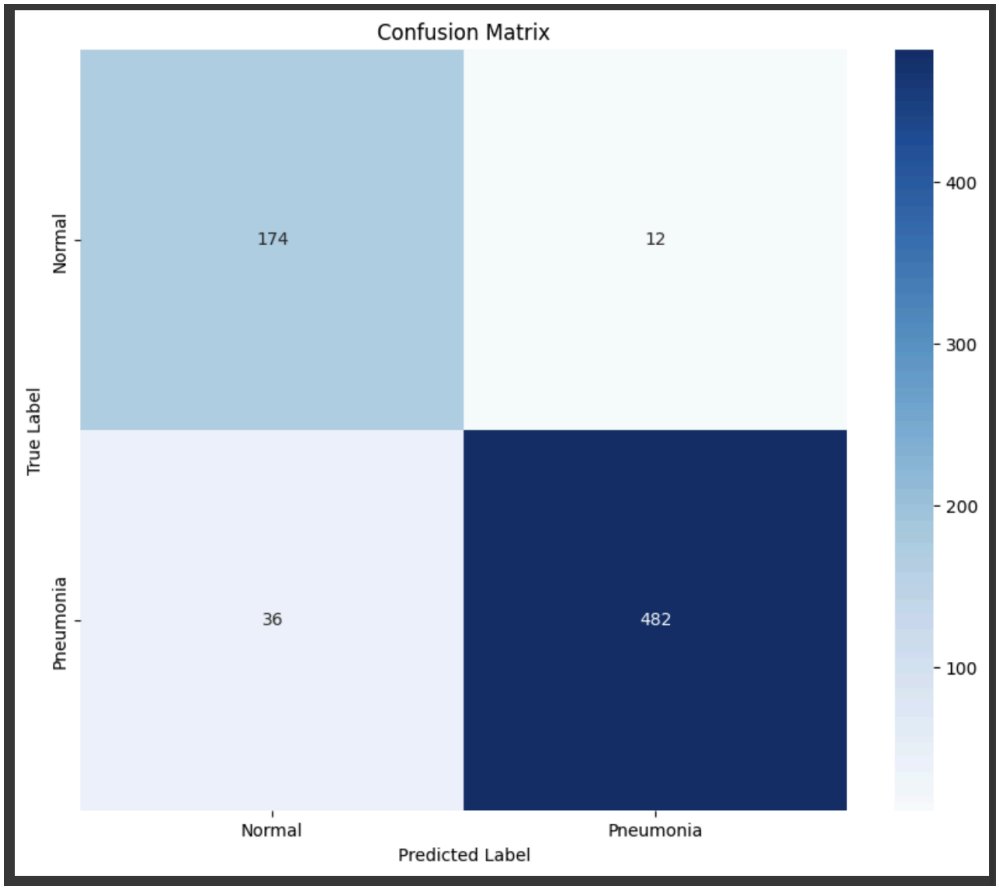
```
Final Training Accuracy: 0.9171
Final Validation Accuracy: 0.9282
```

```
  self.pid = os.fork()
F1 Score: 0.7541
Recall: 0.7437
Precision: 0.7730
```

Visualization:

1) **Training and Validation accuracy:**
   We're plotting the accuracy of validation and training over epochs.

2) **Confusion Matrix:**
   A confusion matrix is plotted between True labels and predicted labels for the classes "Normal" and "Pneumonia".



Improved AlexNet Model:
Architecture:

The improved AlexNet model has reduced the number of convolution layers from 5 (base model) to 3.

The first convolution layer has 32 filters with a kernel size of 11x11 and stride as 4 and padding 2. It is followed by a ReLU activation layer and max pooling.
The second convolution layer has 64 filters with a kernel size of 5x5 and padding 2. It is followed by a ReLU activation layer and max pooling.
The third convolution layer has 128 filters with a kernel size of 3x3 and padding 1. It is followed by a ReLU activation layer and max pooling.

It has 3 linear layers followed by ReLU activation function and dropout layer.

Structure of Improved AlexNet:

```
ImprovedAlexNet(
  (features): Sequential(
    (0): Conv2d(1, 32, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=4608, out_features=1024, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=1024, out_features=512, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=512, out_features=2, bias=True)
  )
)
```

Training and Evaluation:

We're using dropout and early stopping while training the model. Early stopping is used to prevent overfitting of the model. It increases the efficiency of the training process.

```
  warnings.warn("The verbose parameter is deprecated. Please use get_last_lr() "
Epoch 1/20, Train Loss: 0.3080, Train Acc: 0.8712, Val Loss: 0.2522, Val Acc: 0.8918
Epoch 2/20, Train Loss: 0.2459, Train Acc: 0.9000, Val Loss: 0.2535, Val Acc: 0.8850
Epoch 3/20, Train Loss: 0.2177, Train Acc: 0.9141, Val Loss: 0.1998, Val Acc: 0.9271
Epoch 4/20, Train Loss: 0.1989, Train Acc: 0.9249, Val Loss: 0.2102, Val Acc: 0.9237
Epoch 5/20, Train Loss: 0.2134, Train Acc: 0.9141, Val Loss: 0.2083, Val Acc: 0.9180
Epoch 6/20, Train Loss: 0.1908, Train Acc: 0.9285, Val Loss: 0.1779, Val Acc: 0.9237
Epoch 7/20, Train Loss: 0.1862, Train Acc: 0.9268, Val Loss: 0.2327, Val Acc: 0.9226
Epoch 8/20, Train Loss: 0.2045, Train Acc: 0.9197, Val Loss: 0.2109, Val Acc: 0.9180
Early stopping.
Training complete in 10m 6s
```

Metrics:

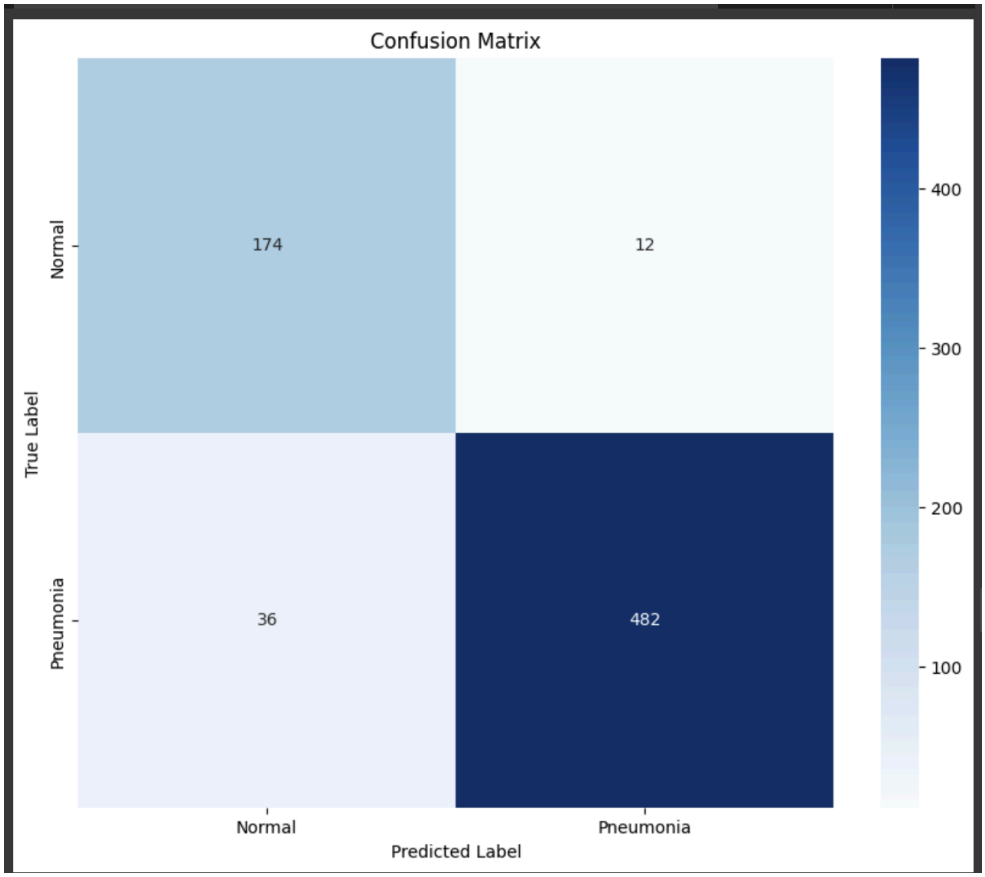
```
Final Training Accuracy: 0.9197
Final Validation Accuracy: 0.9180
```

```
  self.pid = os.fork()
F1 Score: 0.7473
Recall: 0.7358
Precision: 0.7689
```

Visualizations:

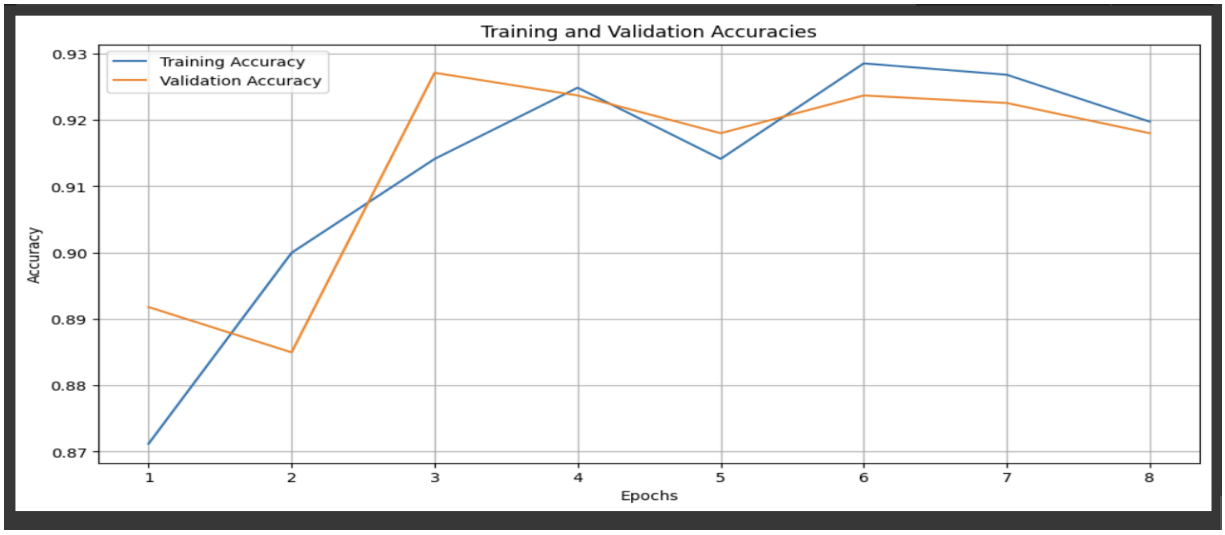
1) **Confusion Matrix:**

A confusion matrix is plotted between True labels and predicted labels for the classes “Normal” and “Pneumonia”.



2) **Training and Validation Accuracies:**

The graph between training and validation accuracies and plotted across epochs.



3) **Training and Validation Loss:**
The graphs between validation and training loss is plotted across epochs.

Model Visualisation using TensorBoard:

TensorBoard is used to make the deep learning models easier by allowing the visual representations of designs, metrics, graphs, images, and how model weights change over time.
It helps us in understanding how the data is prepared before it is integrated into the model.

During training, validation and testing the model, the visualizations continuously monitor and display key metrics such as loss, recall, accuracy etc.

The TensorBoard consists of time series, scalars, images, graphs, PC curves etc.

We get to see and analyze the SummaryWriter of different writers for different projects and help us to compare them and their performance.

Installing TensorBoard:

We use !pip install tensorboard to install tensor board to our code.

Creating a Directory to store tensorBoard Logs:

```
# Creating a directory to store TensorBoard logs
log_dir = './alexnetlogs'
# Initialize SummaryWriter
writer = SummaryWriter(log_dir=log_dir)
```

The logs will be saved at alexnetlogs.

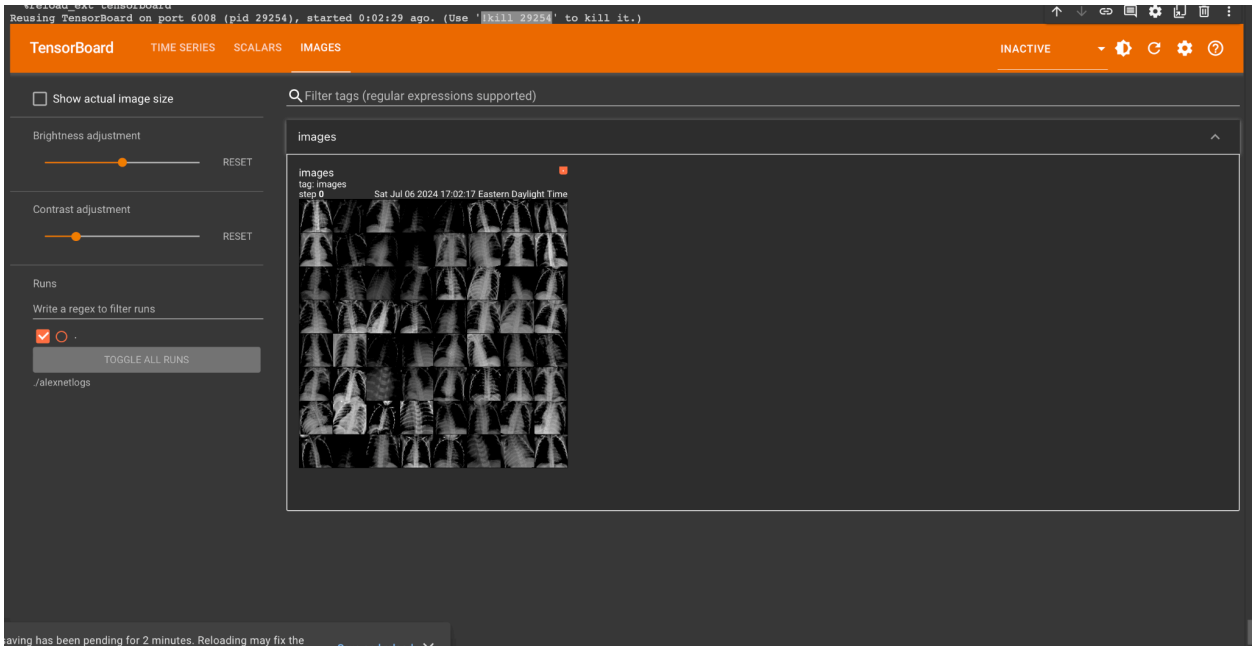
Training the model for Tensor Board:

```
  self.pid = os.fork()
Epoch 1/8, Train Acc: 0.9358, Val Acc: 0.9282
Epoch 2/8, Train Acc: 0.9322, Val Acc: 0.9328
Epoch 3/8, Train Acc: 0.9261, Val Acc: 0.9226
Epoch 4/8, Train Acc: 0.9375, Val Acc: 0.9248
Epoch 5/8, Train Acc: 0.9410, Val Acc: 0.9408
Epoch 6/8, Train Acc: 0.9307, Val Acc: 0.9317
Epoch 7/8, Train Acc: 0.9412, Val Acc: 0.9043
Epoch 8/8, Train Acc: 0.9317, Val Acc: 0.9442
```

Plotting Training and Validation Accuracy:

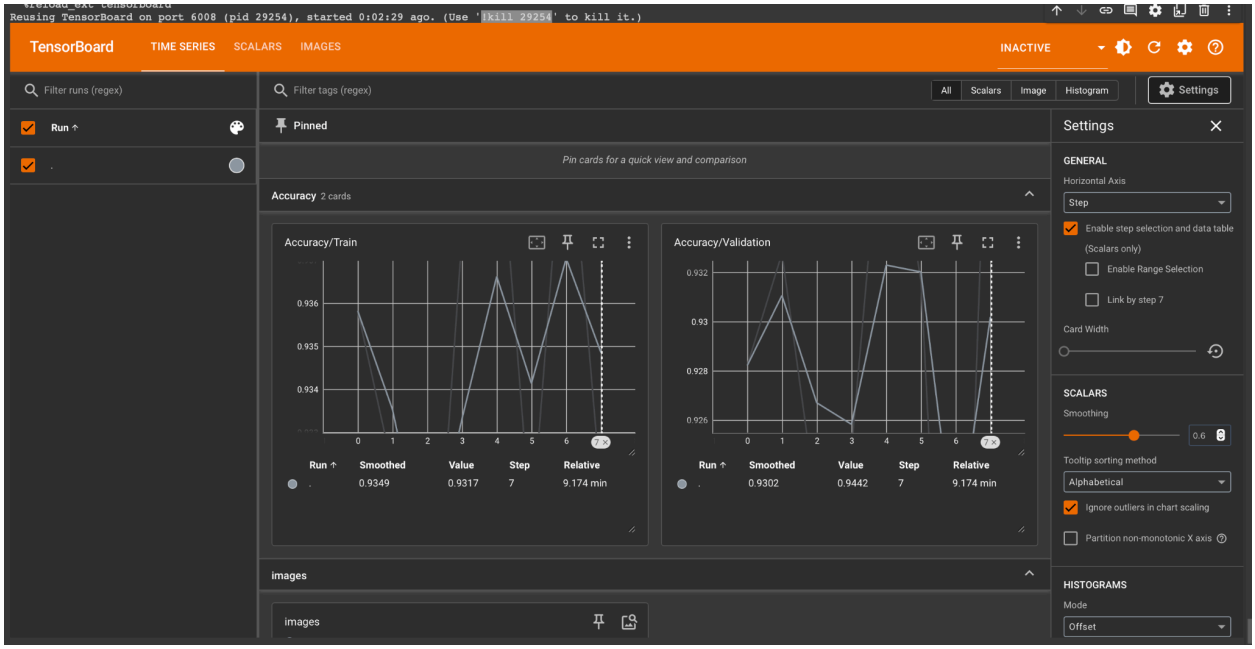
Features visualized in the Tensor Board:

1) **Images:**
   A few images from the dataset are visualized in the Tensor board.

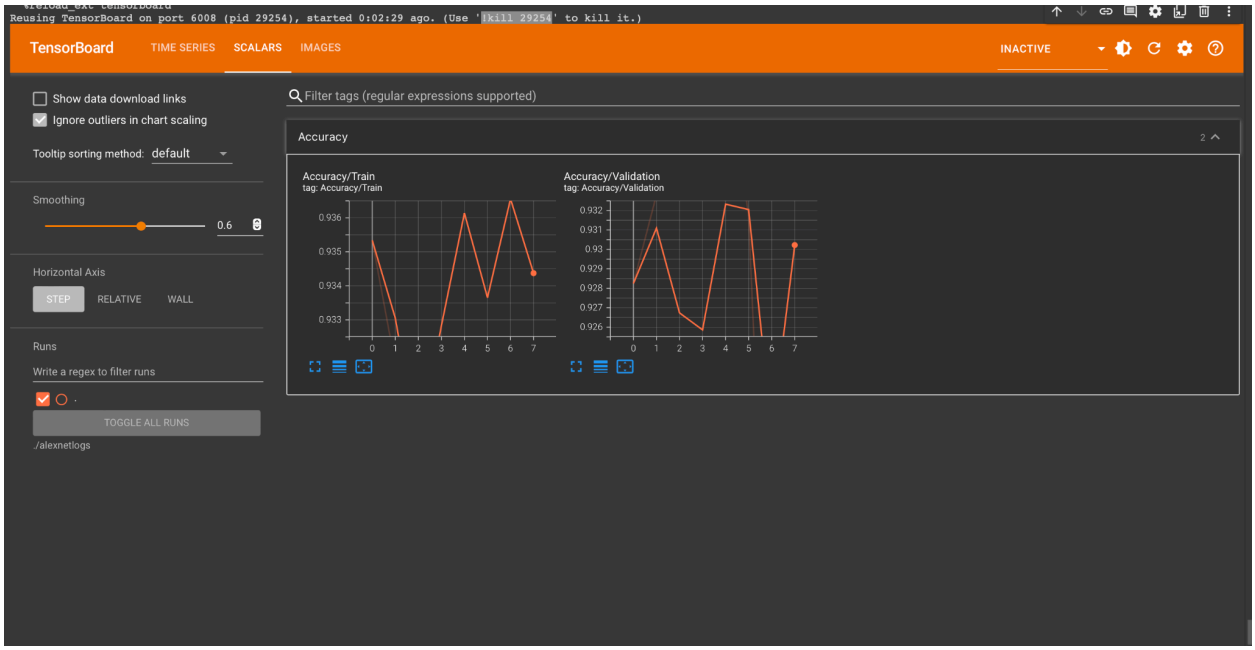


2) **Time Series:**
   The time series is used to visualize metrics of the model performance over

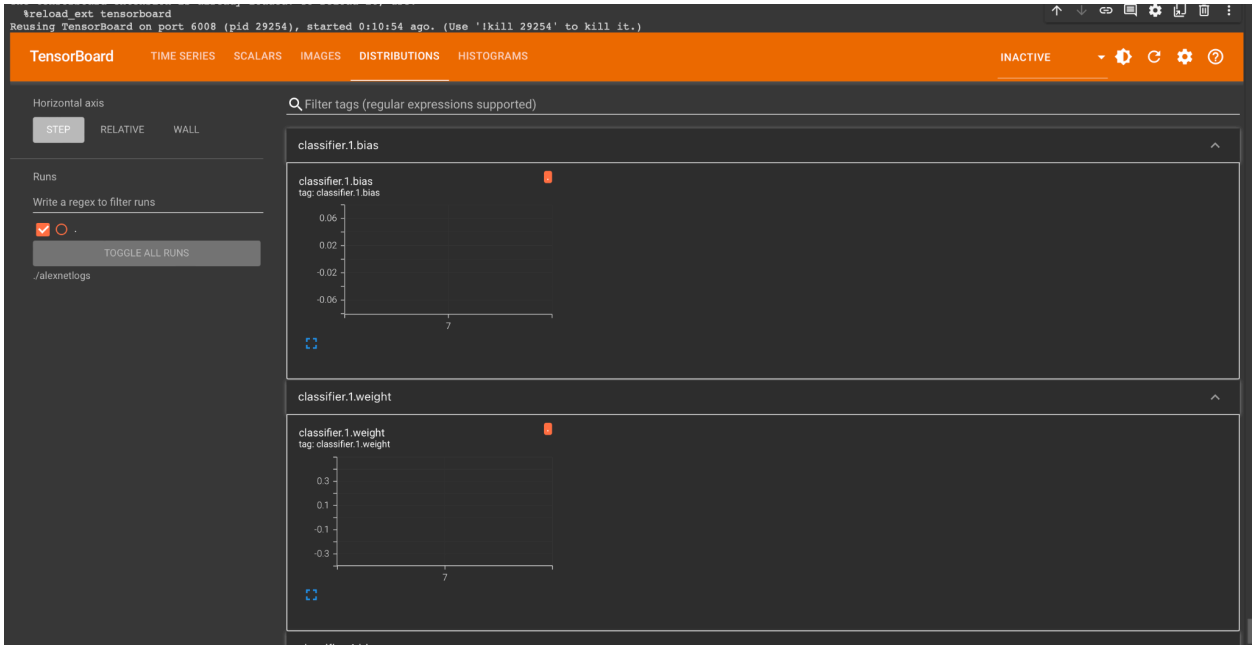
epochs. We can track the changes that are being made live during training the model.It helps us to understand the training of the model.



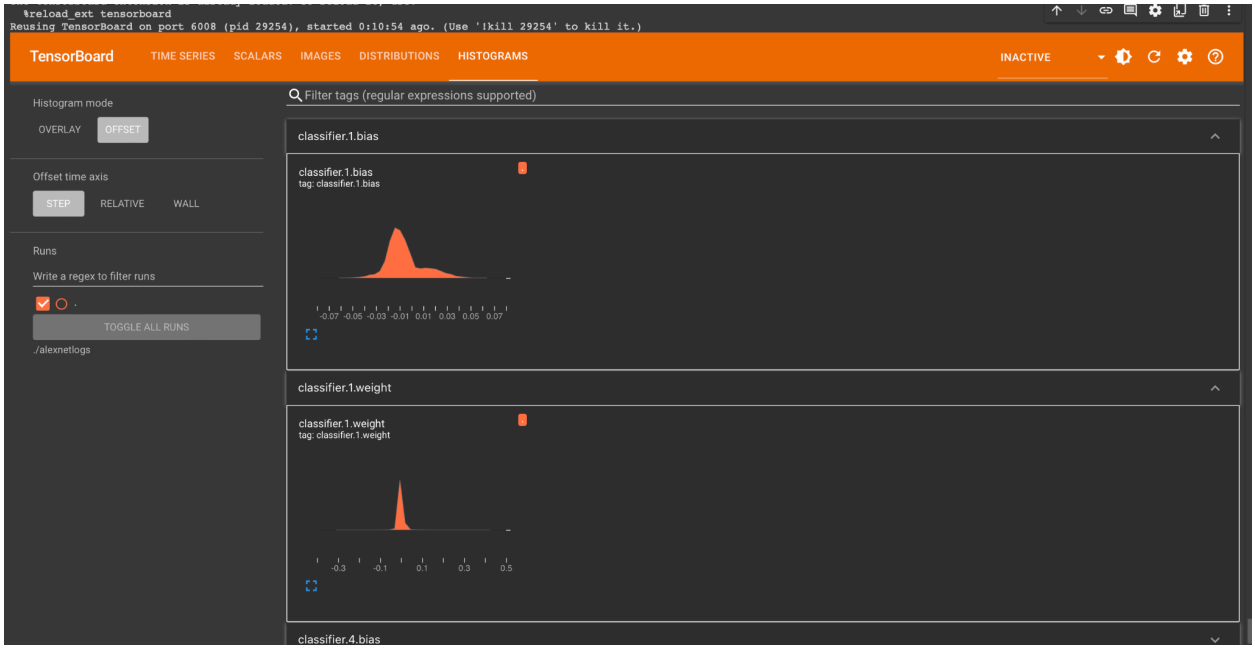
3) **Scalars:**
We shall be able to view training accuracy and validation accuracy being plotted over time. They provide the real time view on how these metrics perform during training of the model. This helps us to understand the model and can be used to make necessary changes.



4) **Distributions:**
We can see the distribution of the tensor values over time during the training of the model. We can see how values such as weights, bias etc change and vary while training the model.



5) **Histograms:**
It provides the histogram image of the tensor values that change over time. They give us a detailed explanation on how the values evolve over different layers.



Conclusion:

On using model visualization using TensorBoard, we get to analyze performance during the training and validation of the model. Through the metrics such as accuracy and loss, we get to learn the live performance while plotting them over epochs. We can have a better understanding of the model and its layers and how metrics like activation function, dropout layer, loss function, optimiser affect the model's performance.

Inferences and Analysis:

By evaluating all the models so far, we can find the below analysis.

Models	**CNN**	**VGG19**	**DenseNet**	**GoogleNet**	**ResNet34**	**AlexNet**
Accuracy	92.84	75.88%	92.04%	95.11%	86.80%	92.82%
Precision	0.92	0.93	0.92	0.95	0.88	0.74
Recall	0.92	0.92	0.92	0.95	0.87	0.77
F1 Score	0.92	0.92	0.92	0.95	0.85	0.75

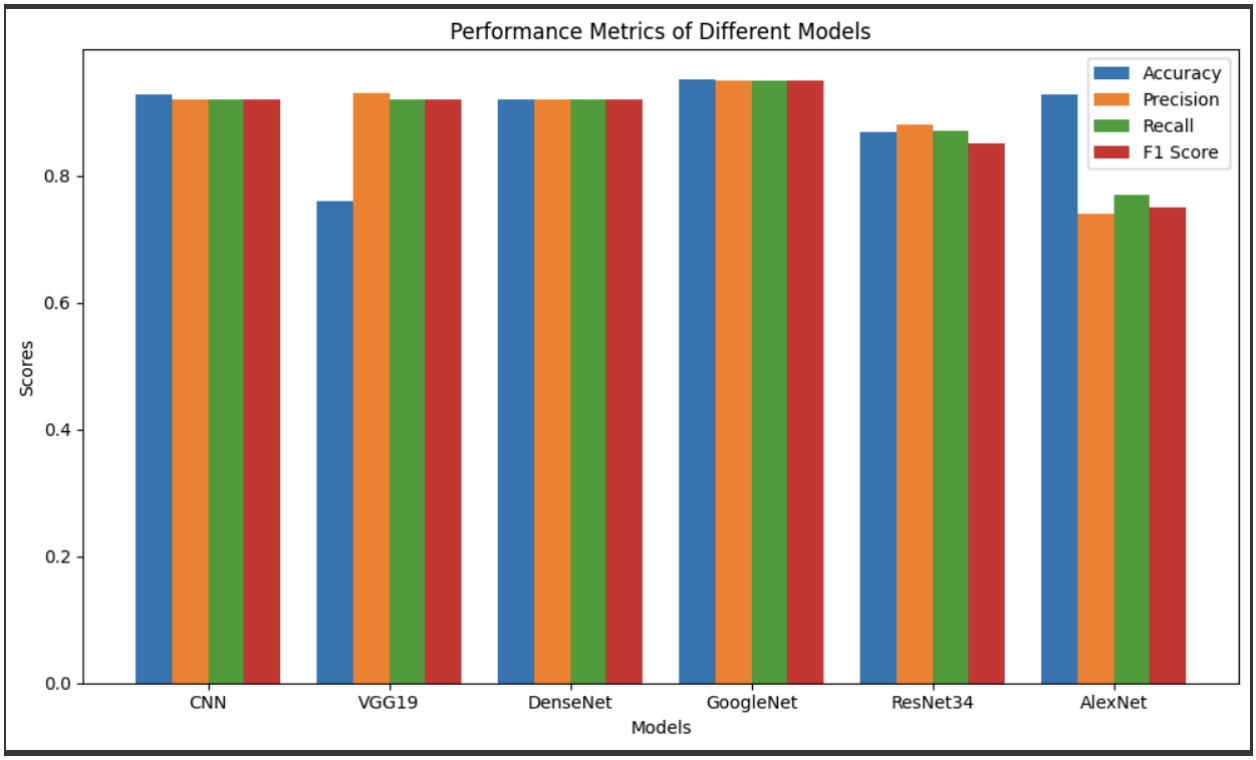
On applying GAN:

Comparative analysis of GAN based images and original dataset

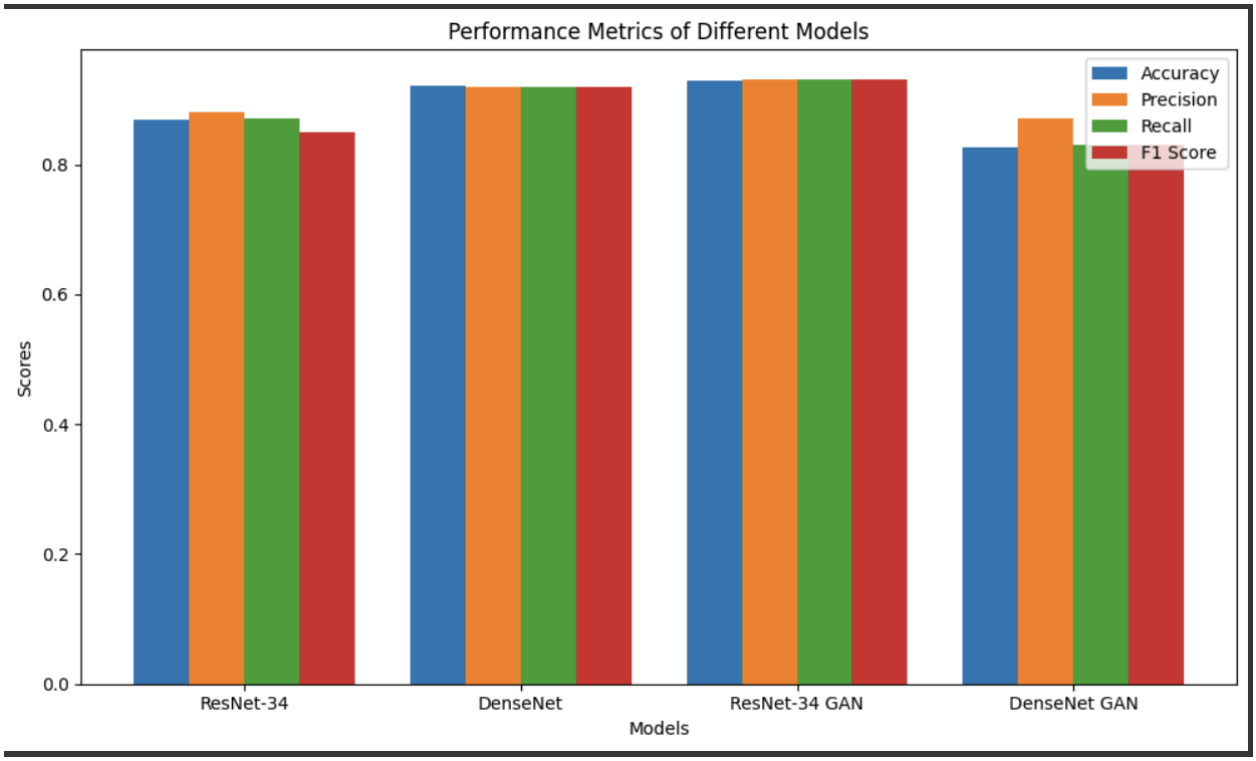
	RESNET-34	**DENSENET**	**RESNET-34 - GAN**	**DENSENET - GAN**
Accuracy	86.80%	92.04%	92.87%	82.66%
Precision	0.88	0.92	0.93	0.87
Recall	0.87	0.92	0.93	0.83
F1 score	0.85	0.92	0.93	0.83

On applying Autoencoders:
Here are the metrics on applying CNN for the test data

```
Test Loss: 0.2966, Test Accuracy: 89.95%
Precision: 0.9064, Recall: 0.8995, F1 Score: 0.9014
```


Performance Metrics of Different Models
Accuracy
Precision
Recall
F1 Score
Scores
0.0
0.2
0.4
0.6
0.8
CNN
VGG19
DenseNet
GoogleNet
ResNet34
AlexNet
Models

For GAN:


Performance Metrics of Different Models
Accuracy
Precision
Recall
F1 Score
Scores
0.0
0.2
0.4
0.6
0.8
ResNet-34
DenseNet
ResNet-34 GAN
DenseNet GAN
Models

The Chest X-Ray Images (Pneumonia) dataset contains **5,856** chest radiograph images in PNG format, categorized into 2 classes: Normal and Pneumonia. It aims to help us in the development of diagnostic models for pneumonia.

The models that we've applied are useful for checking the performance of the dataset. CNN allowed us to enable deeper networks without letting the gradients disappear, whereas VGG-19 has an increased training time with the least accuracy when compared to other models.

From ResNet, we can identify that due to the residual connections, the model performed without losing the gradients, but the accuracy is less. GoogleNet has the highest accuracy with the layers reducing the complexity of the model.

DenseNet can reuse the features across the layers which allows it to contribute to high accuracy and performance. On the other hand, AlexNet doesn't have the complexity to work on complex datasets.

On applying GAN, we've generated images that are introduced into the dataset. On introducing these images, we can see that DenseNet performs better without the GAN images while densenet performs worse when the images are introduced .

Whereas the Resnet's ability to skip connections might be better at handling the characteristics of GAN- generated images and they might be more resilient to any artifacts or slight inconsistencies in the GAN generated images.

The autoencoder successfully decreased dimensionality by retaining crucial features and reducing computational load. Incorporating these encoded images for training reduced training time still having excellent performance. The modified CNN model is applied with the encoded images. It has attained a validation accuracy of 90.77% and training time of just 17.52 seconds.

Conclusion:

In conclusion, we have taken the Pneumonia dataset and performed various deep learning models, some simple models like CNN, and some complex models like ResNet, DenseNet.

Overall, we can find that complex models like Resnet, DenseNet, GoogleNet have had better performance when compared to simple CNN models like Base CNN, VGG-19, AlexNet in terms of comparing the metrics like accuracy, precision, recall, f1 score.

On top of it, we have applied complex models like ResNet and DenseNet to the dataset which has GAN generated images. On evaluating the performance, we can see that DenseNet performed well without the GAN images while ResNet has accomplished a better overall performance with GAN images introduced.

When Autoencoders were applied on the dataset as a preprocessing step, we can see that the dimensionality of the images were decreased and the execution time taken for CNN model to those images were very less and had a high accuracy.

With these models and evaluation, we will be able to apply these deep learning models on chest X-rays image data to classify whether a person suffers from pneumonia or not. This method improved detection and diagnosis of patients who might have pneumonia and further helps the physicians in treating the patients better enhancing the health outcomes.

Real-world deep learning application [3 points]

When we're integrating the deep learning models with the health-care system in the real world, they can give us the real-time analysis of the X-ray images to check whether the patients have pneumonia or not, which will help doctors to provide a proper diagnosis on the health condition of the patient.
These models can be combined with the X-ray machines, so when an x-ray is taken, it can help in the analysis and can speed up the diagnosis if the patient is suffering from pneumonia.
This will benefit patients suffering from pneumonia to receive appropriate treatment.

References:

1)https://www.analyticsvidhya.com/blog/2021/03/introduction-to-the-architecture-of-alexnet/
2) https://thecleverprogrammer.com/2021/12/13/alexnet-architecture-using-python/
3) https://www.w3schools.com/python/python_ml_train_test.asp
4) https://www.youtube.com/watch?v=VJW9wU-1n18&t=460s
5) https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
6) https://www.youtube.com/watch?v=VJW9wU-1n18&t=460s
7) https://scikit-learn.org/stable/modules/neural_networks_supervised.html
8) https://pytorch.org/tutorials/beginner/saving_loading_models.html
9) https://ar5iv.labs.arxiv.org/html/2107.02970
10) Deep Learning (2017 edition)– Ian Goodfellow, Yoshua Bengio, Aaron Courville
11) https://www.wallstreetmojo.com/generative-adversarial-network/
12) https://www.clickworker.com/ai-glossary/generative-adversarial-networks/

13)https://ai.stackexchange.com/questions/25601/what-are-the-fundamental-differences-between-vae-and-gan-for-image-generation
14) https://resources.pcb.cadence.com/blog/2023-gan-advantages-and-disadvantages
15) https://www.simplilearn.com/tutorials/deep-learning-tutorial/generative-adversarial-networks-gans
16) https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b
17) https://d2l.ai/chapter_computer-vision/transposed-conv.html
18) https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11
19) https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc
20)https://medium.com/analytics-vidhya/downsampling-and-upsampling-of-images-demystifying-the-theory-4ca7e21db24a
21)https://dsp.stackexchange.com/questions/71685/upsampling-vs-downsampling-which-to-use-when#:~:text=Downsampling%20reduces%20dimensionality%20of%20the,the%20resolution%20of%20previous%20layer
22) Class Notes
23) Deep Learning (2017 edition)– Ian Goodfellow, Yoshua Bengio, Aaron Courville
24) https://www.geeksforgeeks.org/auto-encoders/
25) https://plotly.com/python/line-charts/
26) https://www.v7labs.com/blog/autoencoders-guide
27) https://www.mdpi.com/2227-7390/11/8/1777

28) https://arxiv.org/abs/2003.05991
29) https://www.youtube.com/watch?v=EujccFRio7o

Trello DashBoard: