CMP6228
Deep Neural Network
Chest X-Ray Images Pneumonia Detection
Course: Computer and Data Science

Student Name: Habban Islam
Student Number: 20109816

Contents

1 Abstarct

This report will include discussion of several deep learning model which can be used for chest x-ray pneumonia detection. Three models have been implemented from scratch and four pre-trained models been used ResNet50, EfficientNetB3, VGG19, and DenseNet 169. Before implementing these models, dataset have been pre-processed to make it ready for the models. Finally, to check models performance various evaluation metrics have been used like accuracy, recall, precision, f1-score.

2 Introduction

(Shortliffe, E.H. and Blois, M.S., 2006) stated that the arrival of digital computers in 1940s prompted people to learn about their future usefulness in medicine, and ever since then, several technical developments on it have made computer usage unavoidable in everyday healthcare. However, the healthcare industry has taken its time to completely grasp and integrate information technology into the workplace. Nowadays, thereâs a lot of attention being given to the development of electronic health records, which is essential for planning and having a better knowledge of patience and looking for ways to improve. Paper-based medical records are still used, although it's becoming more and more ineffective and irritating as it is very time-consuming.

Due to the similarity in symptoms between illnesses, it can be difficult for doctors to diagnose lung infections using chest X-rays. Misdiagnosis can result in ineffective therapy and put the patientâs life in danger (Devvret Verma, 2020). He also talked about how different framework is used for classifying TB, pneumonia etc in chest X-rays using a neural network classifier.

This project aims to create a deep-learning model which can identify pneumonia from X-ray images. It will consider rescaling, shifting and different angles of images to maximise its learning.

The selected dataset for this project is from Kaggle, originally its been taken from Cell. It contained total of 5863 chest x-ray images, 1583 normal and 4273 pneumonia chest x-ray images. Chest X-ray images were from paediatric patients aged one to five from Guangzhou Women and Childrenâs Medical Centre. All chest radiographs were originally checked for quality control by removing low-quality and unreadable scans from the study. Before the diagnosis for the images could be used to train the deep learning model, they were assessed by two experienced doctors. A third expert also reviewed the assessment set to make sure there were no grading mistakes.

3 Problem Statement

When it comes to identifying and treating lung illnesses like pneumonia, chest X-rays are a crucial diagnostic tool in medicine. Yet, because many of the illnesses can resemble one another and there is potential for misinterpretation, it can be difficult for medical professionals to read these chest X-rays. By offering an automatic and more accurate interpretation of chest X-rays, deep learning can assist in solving this problem. Deep learning algorithms can be trained to recognise patterns in chest X-rays that are connected to certain illnesses by employing neural network classifiers. This will provide better patient outcomes as doctors will be able to diagnose patients more precisely and select the best course of therapy.

Deep learning can also swiftly and reliably handle massive volumes of medical data, enabling quicker and more accurate diagnosis. By making healthcare delivery more effective, it has the potential to

both increase access to healthcare and lower healthcare expenditures.

These are a few steps which can be followed to use deep learning algorithms to detect pneumonia accurately from X-ray images:

1. Create a deep learning algorithm that can successfully identify pneumonia in X-ray pictures.
2. Make sure the algorithm can identify pneumonia from other respiratory illnesses that can resemble it on X-ray pictures.
3. Reach at least 90% accuracy rate in X-ray image-based pneumonia detection.
4. Lower the algorithm's false positive and false negative rates.

These are few aims of the project and how it can be met (using SMART):

Specific

- Create a deep learning algorithm CNN that can successfully identify pneumonia in X-ray images
- Ensure that the CNN model can identify pneumonia, normal and other lung diseases that can resemble from X-ray images.

Measurable

- Reach at least 90% accuracy rate in identifying pneumonia using CNN
- Lower the model's false positive and false negative rates to reduce cost and avoid putting patients in danger

Achievable

- Get a big dataset of X-ray images that have labelled and unlabelled images showing pneumonia or not
- Do EDA and pre-processing to the dataset to get more insight and make it ready for training
- To improve models performance; train, test and validate the dataset
- Use the evaluation matrix to assess and improve the model and make any changes to its parameter accordingly

Relevant

- Pneumonia is a frequent and deadly lung infection that, if not identified and treated promptly, can be fatal
- Doctors can make a more accurate diagnosis and provide better care by accurately identifying pneumonia from X-ray images

Time-bound

- Regularly assess the performance of the model and make any required modifications to it if the need arises.

4 Proposed Method

For the proposed model different CNN models will be looked at as other deep learning architecture like RCNN, YOLO, SSD not as effective and evaluated using confusion matrix to see their performances. For the model training there are steps such as Data collection, data pre-processing, model training, model deployment, model improving, model evaluation and model monitoring. Model improving and evaluation can be repeated until suitable model is found. The concept solution diagram is provided below:

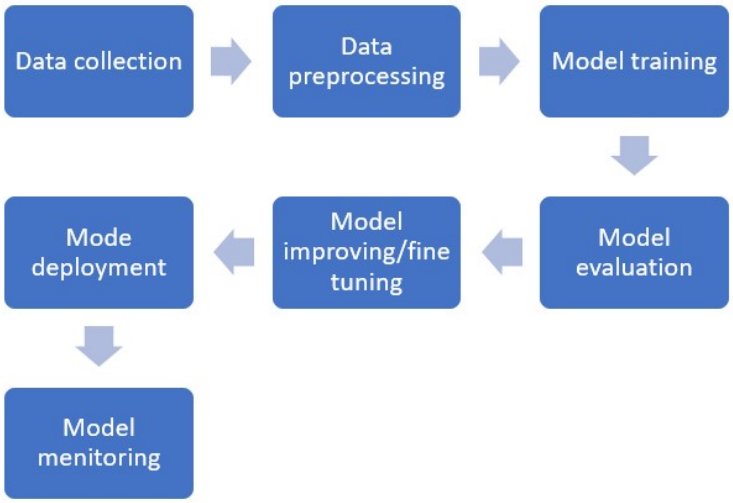


Figure 1: Concept Solution

4.1 Data Loading and Pre-processing

```
[ ]  1  labels = ['NORMAL','PNEUMONIA']
```

```
1  dataset = []
2  label = []
3
4  image_size = 224
5  print("DataSet")
6  print("============")
7  for i in labels:
8      folderPath = os.path.join('/content/drive/MyDrive/DNN/Chest_X-ray/BT_Dataset_without_DA',i)
9      for j in tqdm(os.listdir(folderPath)):
10         # reading the images using cv2 library
11         img = cv2.imread(os.path.join(folderPath,j))
12         # Image resizing
13         img = cv2.resize(img,(image_size, image_size))
14         dataset.append(img)
15         label.append(i)
16
17 dataset = np.array(dataset)
18 label = np.array(label) #.reshape((-1, 1))
```

```
DataSet
============
100%|██████████| 1583/1583 [00:41<00:00, 38.26it/s]
100%|██████████| 4273/4273 [00:42<00:00, 100.84it/s]
```

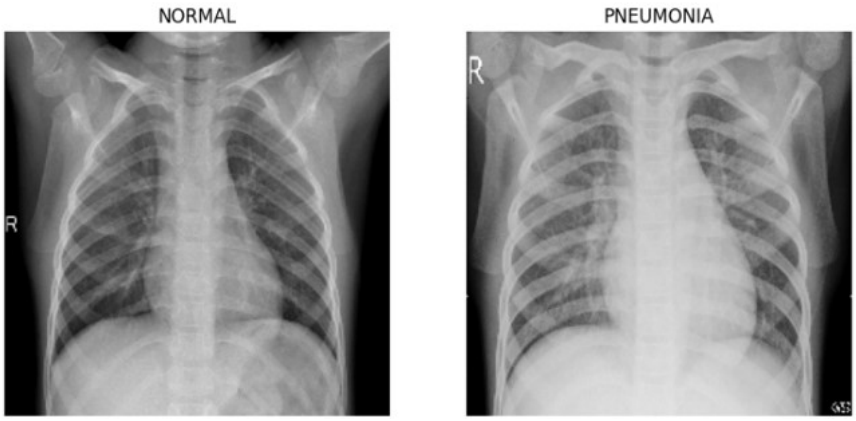
Figure 2

The code reads and resizes images from the â/content/drive/MyDrive/Project/DNN/ BT_Dataset_without_DA" directory's two folders with the labels "NORMAL" and "PNEUMONIA." The photos are resized to a square shape with 224*224 pixel dimensions using the

OpenCV framework. The resized photos are kept in a list called a "dataset," and the labels for those images are kept in a list called a "label." 'PNEUMONIA' denotes images with pneumonia, whereas 'NORMAL' denotes images without pneumonia. The label lists and dataset are then converted to NumPy arrays for further processing.

Plot Sample Images

```
[ ]  # Displaying sample images from bith classes
     k=0
     fig, ax = plt.subplots(1,2,figsize=(10,10))

     for i in labels:
         j=3
         while True :
             if label[j]==i:
                 ax[k].imshow(dataset[j])
                 ax[k].set_title(label[j])
                 ax[k].axis('off')
                 k+=1
                 break
             j+=1
```



This is a sample image with ânormalâ and âpneumoniaâ images from the dataset.

Data Shuffling & Normalization

```
     dataset, label = shuffle(dataset,label, random_state=101)
```

```
     # normalize dataset
     dataset = dataset.astype('float32')
     dataset /= 255
```

```
[ ]  print("dataset=",dataset.shape)
     print("label=",label.shape)
```

```
dataset= (5856, 224, 224, 3)
label= (5856,)
```

The shuffle () function from the scikit-learn package is used to randomly shuffle the dataset and label arrays, with a random_state value of 101 preventing the training data from being impacted by the loading order of the images. The dataset array is shuffled, converted to a NumPy array with a float32 datatype, then normalised by dividing each pixel's value by 225 to scale it to fall between 0 and 1. Normalisation is frequently used to boost neural networks' functionality and capacity.

```
#Spliting dataset 80% for training and 20% for testing
X_train,X_test,y_train,y_test=train_test_split(dataset,label,test_size=0.2,random_state = 3, shuffle=True)
X_train = normalize( X_train, axis =1)
X_test = normalize( X_test, axis =1)
print("X train=",X_train.shape)
print("Y train=",y_train.shape)

print("X test=",X_test.shape)
print("Y test=",y_test.shape)
```

```
X train= (4684, 224, 224, 3)
Y train= (4684,)
X test= (1172, 224, 224, 3)
Y test= (1172,)
```

The dataset and label arrays are divided into training and testing sets using the train_test_split () function from the scikit-learn package, with a test_size parameter value of 0.2 indicating that 20% of the data will be utilised for testing and the remaining 80% for training. To guarantee that the same split is produced each time the code is executed, the rand0m_state argument is set to 3. Before splitting, the data are shuffled with the Shuffle parameter set to True, which helps to eliminate bias that may be induced by the order of the data. The X_train and X_test arrays images pixel values are normalised along the given axis, which is set to 1, using the normalise () function. After the data has been split, this is done. Each row of images is thus normalised independently. Normalisation improves the neural network's performance during training by guaranteeing that the input data has a mean of 0 and a standard deviation of 1.

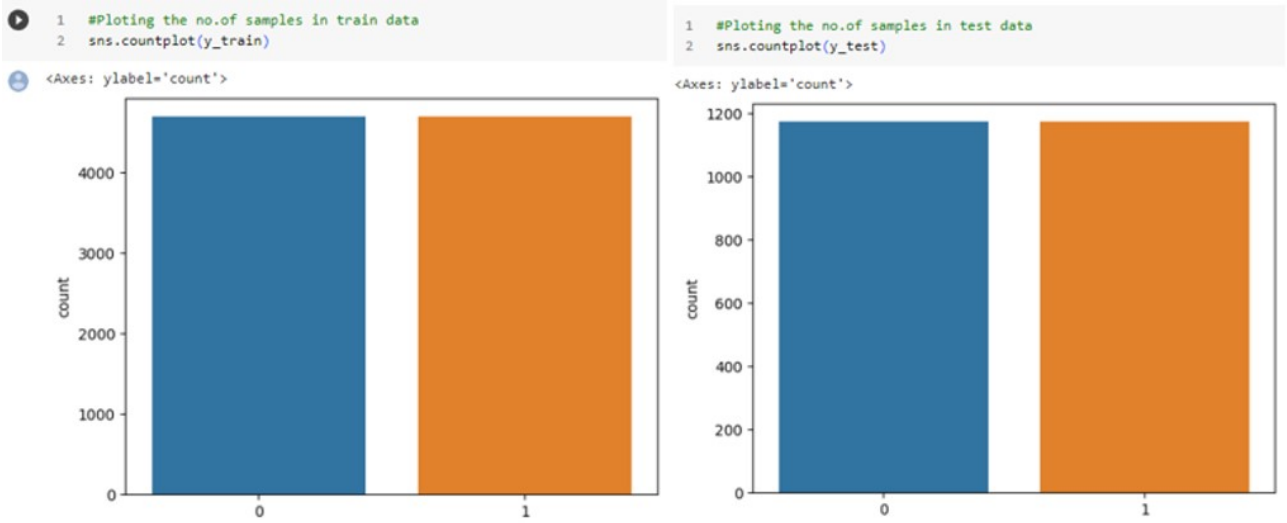
Label Encoding

```
# label encoding - one hot encoding

y_train_new = []
for i in y_train:
    y_train_new.append(labels.index(i))
y_train = y_train_new
y_train = tf.keras.utils.to_categorical(y_train)


y_test_new = []
for i in y_test:
    y_test_new.append(labels.index(i))
y_test = y_test_new
y_test = tf.keras.utils.to_categorical(y_test)
```

```
#Ploting the no.of samples in train data
sns.countplot(y_train)
```

The labels in the y_train and y_test arrays, "NORMAL" and "PNEUMONIA," are transformed from their string format to an encoded format that is suitable for training a neural network. This is accomplished by using the to_categorical () method of the TensorFlow Keras API. First, a new list called y_train_new is created and contains the encoded labels for the y_train array. The index () function is used to find the index of each label in the labels list. The index of each label is subsequently added to the y_train_new list. Then the encoded labels are added to the y_train array by setting it

equal to y_train_new. The _categorical () function is then used to convert the encoded labels into one-hot encoded format. To represent each label as a vector of binary values, the corresponding index for each label is set to 1, and all other indexes are set to 0. Since it enables the network to forecast a probability distribution across all probable labels, this format is frequently used to train neural networks. The y_test array underwent the same procedure for one hot encoding of testing labels.



There are 1172 pneumonia and normal images in the testing set compared to 4684 pneumonia and normal photos in the training set, according to the count plot above. Therefore, the collection has 5856 chest X-ray pictures in total. As the dataset relatively small, data augmentations may be used to enhance the number of the images and improve the performance of the models.

4.2 Implementation

Three CNN models were built from scratch for detection. Each of them is an enhanced version of the one before it, and the final third of them all performed the best. Four pre-trained models have also been implemented.

4.2.1 Model 1

32 neurons filters of size 3x3 are contained in the first convolutional layer. The input_shape option specifies the 224x224 pixel, three RGB colour channel shape of the input pictures. With the use of the relu activation function, the layer's output was rendered non-linear. This is due to real-world issues are rarely linear, which makes relu a strong choice for simulating intricate patterns in data. Due to the max pooling layer's pool size of 2x2, the spatial dimensions of the output feature maps are reduced by a factor of 2. As a result, the model's parameter count was reduced to prevent overfitting.

Everything remained the same for the second and third layers, but the number of neurons was increased to 64 and 128 correspondingly, allowing the model to extract additional spatial information from the input pictures. Then it was reduced one more to avoid overfitting.

The layer flatten function converts the output of the third convolutional layer into a 1D feature vector, which is subsequently transmitted to the fully connected layer. 64 units plus a relu make up the first fully linked layer, which uses all the attributes that have been learnt from earlier layers to create predictions. Two units make up the last fully linked layer, which uses the sigmoid activation

Model 01 - CNN model from scratch with Three Layers

```
# Define the CNN model architecture
model = tf.keras.Sequential([
    #Layer 1
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(224, 224, 3)),
    layers.MaxPooling2D((2,2)), # I will write here about the detail of max pooling
    #Layer 2
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    #Layer 3
    layers.Conv2D(128, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    #FC 1
    layers.Dense(64, activation='relu'),
    #FC 2
    layers.Dense(2, activation='sigmoid')
])
```

```
model.summary()
```

```
# Compile the model with binary cross-entropy loss and Adam optimizer
model.compile(optimizer=Adam(learning_rate=1e-4), loss='binary_crossentropy', metrics=['accuracy'])
```

```
checkpoint = ModelCheckpoint("/content/drive/MyDrive/DNN/Chest_X-ray/Implementation/Saved_Models/Model1_CNN.h5",
                             monitor="val_loss",save_best_only=True,mode="auto",verbose=1)
reduce_lr = ReduceLROnPlateau(monitor = 'val_loss', factor = 0.3, patience = 5, min_delta = 0.001,
                              mode='auto',verbose=1)
```

Training The Model

```
history = model.fit(X_train,y_train,validation_split=0.1, shuffle = True, epochs = 100, verbose=1, batch_size=32,
                    callbacks=[checkpoint,reduce_lr])
```

function to construct a probability distribution over the two possible classes of "PNEUMONIA" and "NORMAL."


```
# Compile the model with binary cross-entropy loss and Adam optimizer
model.compile(optimizer=Adam(learning_rate=1e-4), loss='binary_crossentropy', metrics=['accuracy'])
```

```
checkpoint = ModelCheckpoint("/content/drive/MyDrive/DNN/Chest_X-ray/Implementation/Saved_Models/Model1_CNN.h5",
                             monitor="val_loss",save_best_only=True,mode="auto",verbose=1)
reduce_lr = ReduceLROnPlateau(monitor = 'val_loss', factor = 0.3, patience = 5, min_delta = 0.001,
                              mode='auto',verbose=1)
```


The Keras compile () function is used to configure the learning process for the model. Adam, an optimisation method with a 1e-4 learning rate, is applied in this case. The step size for each iteration is determined by learning rate, and this can significantly affect how well the model performs. Faster convergence from a higher learning rate can lead to the model being overfit to the training data, whereas slower convergence from a lower learning rate can lead to greater generalisation and better prediction from the model. Convergence in this case indicates that the neural network has absorbed all the information it could from the training data and is no longer able to outperform it. The network determines the ideal set of parameters for minimising the loss function on the training data.

For this binary classification job, which evaluates the discrepancy between the true labels and the predicted probabilities for each class, binary_crossentropy is employed as the loss function.

Accuracy evaluation matrices are used in this model; they quantify the percentage of properly identified samples in the dataset.

Call-back capabilities The ReduceLROnPlateau () function avoids overfitting and enhances the generalisation of the model by lowering the learning rate by a factor of 0.3 when the validation loss stops increasing. ModelCheckpoint () enables the best performing model to be stored for subsequent use. Additionally, it delays lowering the learning rate suggests patience parameter by five epochs. The value of min_delta, which in this example is 0.001, denotes the minimal changes in the monitored quantity required to qualify as an improvement.

Training The Model

```
history = model.fit(X_train,y_train,validation_split=0.1, shuffle = True, epochs = 100, verbose=1, batch_size=32,
                    callbacks=[checkpoint,reduce_lr])
```

The model is trained using the fit () approach using the training data X_train and y_train, where X_train serves as the input and y_train serves as the goal. 10% of the training data were further utilised for validation.

Verbose 1 will output the degree of information during training and use 32 samples for each batch while it shuffles and trains the model for 100 epochs. If the validation loss does not decrease, the callback routines indicated earlier will be utilised to store the best model and lower the learning rate.

After doing the first model, it is noticed that the accuracy is low this might be due to the imbalance data as there were 1583 Normal and 4273 Pneumonia images. So, balance the data between classes 2 fold data augmentation was done on normal images to increase the number of images.

Data Augmentation on Normal class for class balancing

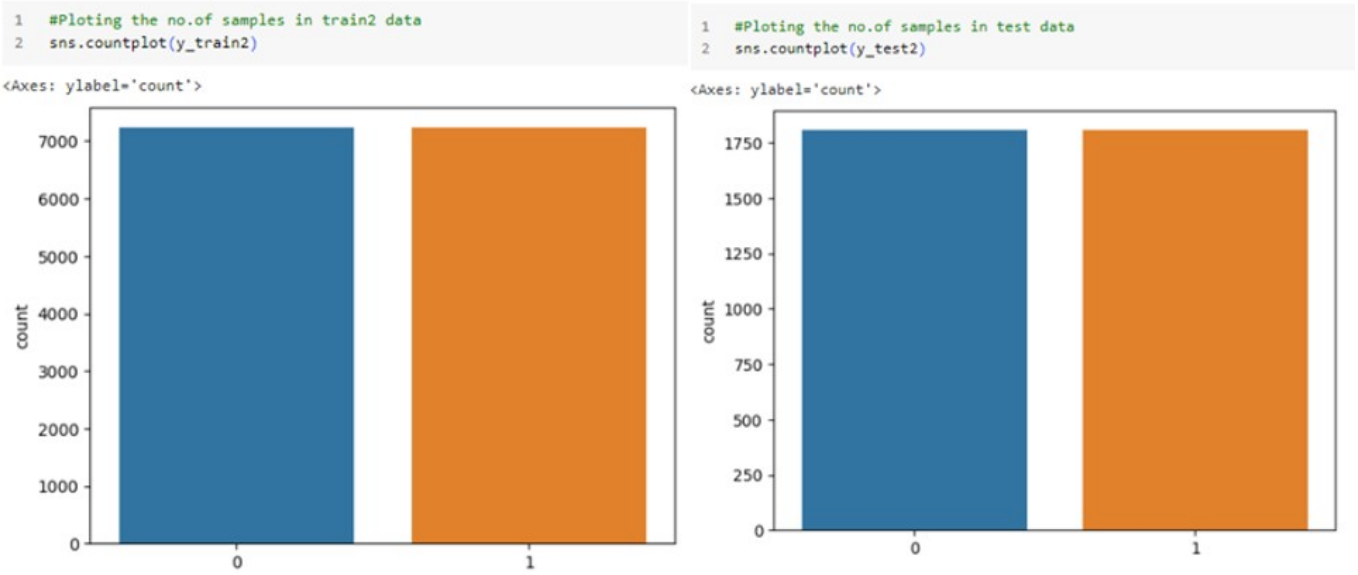
```
# Defining the types of the Data Augmentation
def augment_data(file_dir, n_generated_samples, save_to_dir):
    data_gen = tf.keras.preprocessing.image.ImageDataGenerator(

                                       rotation_range=15,
                                       width_shift_range=0.1,
                                       height_shift_range=0.1,
                                       shear_range=0.1,
                                       brightness_range=(0.3, 1.0),
                                       horizontal_flip=True,
                                       vertical_flip=True,
                                       fill_mode='nearest'
                                      )
    for filename in listdir(file_dir):
        print(filename)
        # load the image
        image = cv2.imread(file_dir + filename)
        # reshape the image
        image = image.reshape((1,)+image.shape)
        # prefix of the names for the generated sampels.
        save_prefix = 'aug_' + filename[:-4]
        # generate 'n_generated_samples' sample images
        i=0
        for batch in data_gen.flow(x=image, batch_size=1, save_to_dir=save_to_dir,
                                   save_prefix=save_prefix, save_format='jpg'):
            i += 1
            if i > n_generated_samples:
                break
```

The function augment_data, which adds data to a group of photos in a specified directory, is defined in this code. Using the Keras ImageDataGenerator class, the function does many types of data

augmentation, such as rotation, shifting, shearing, brightness altering, and flipping. The enhanced images are then stored in a new directory with prefixed jpeg filenames. Three inputs are required by the function; the file_dir, n_generated_samples, and save_to_dir arguments provide the directories in which the input photos are saved and the number of enhanced images that should be generated for each input, respectively. images, Aas well as the location where the created images should be kept. The code-based method iterates over the input directory's files, reading each picture with the OpenCV imread function, resizing it to have a batch size of 1, and then producing n_generated_samples. Enhanced photos created with the ImageDataGenerator class' flow function. When there are more created samples than n_generated_samples, the loop ends.

```
start_time = time.time()
augmented_data_path = '/content/drive/MyDrive/DNN/Brain_Tumor_Detection/BT_Dataset_4DA/'
## These directories contain the images for the two classes 'NORMAL' and 'PNEUMONIA' .
normal_path    = '/content/drive/MyDrive/DNN/Brain_Tumor_Detection/Balanced_aug_dataset/NORMAL/'
pneumonia_path    = '/content/drive/MyDrive/DNN/Brain_Tumor_Detection/Balanced_aug_dataset/PNEUMONIA/'
augment_data(file_dir=normal_path, n_generated_samples=3, save_to_dir=augmented_data_path+'NORMAL')
augment_data(file_dir=normal_path, n_generated_samples=3, save_to_dir=augmented_data_path+'PNEUMONIA')
end_time = time.time()
execution_time = (end_time - start_time)
print(f"Elapsed time: {hms_string(execution_time)}")
```

Finally, this method creates extra augmented photos that are stored to a given directory after performing data augmentation on a series of images of chest x-ray. When there is a 'NORMAL' label on a case, the augment_data function is run once, and when there is a 'PNEUMONIA' label, it is called twice. The time it took the code to run was also computed and shown. Each model that are implemented afterwards used this augmented dataset. The process for the models is the same; the only difference is that the dataset was changed to new enhanced picture data, allowing the same models to be performed with more photos, which enhances model performance. Additionally, certain models now have more input data. To make the model work properly and effectively extract characteristics from the additional photos, several modifications were performed, such as adding new layers and increasing the number of neurons. The evaluation section will explain the results of these and how data augmentations impacted model performance.



Visualisation of the train and testing datasets following a 2 fold augmentation on the normal class of the dataset, which adds 2 more photos for each image in the normal class. The dataset had 9038 X-ray images, 4765 images of normal tissue and 4273 images of pneumonia, divided into 7230

training images and 1808 testing images.

Model 1 was repeated with new augmented dataset to see if there is any improvement. After finding out that it has improved each model after that has used the augmented dataset.

4.2.2 Model 2

Model 02 - CNN model from scratch with Four Layers

```
# Define the input shape
input_shape = (224, 224, 3)

# Define the model
cnn_model = tf.keras.Sequential([
    # Convolutional layer 1
    layers.Conv2D(16, (3,3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2,2)),
    layers.BatchNormalization(),

    # Convolutional layer 2
    layers.Conv2D(32, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.BatchNormalization(),

    # Convolutional layer 3
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.BatchNormalization(),

    # Convolutional layer 4
    layers.Conv2D(128, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.BatchNormalization(),

    layers.Flatten(),

    # FC Layer
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.5),
    layers.BatchNormalization(),

    layers.Dense(2, activation='sigmoid')
])
```

The network learns basic and general properties that are useful for identifying patterns in the input data by beginning with a lower number of filtersâ16 in the first layer. As the network goes deeper and has more sophisticated and precise features that are important for detecting pneumonia, filter size rises for other layers to 32, 64, and 128. Relu and max pooling have been introduced for each layer. The BatchNormalization () layer normalises the output of each layer to boost the model's stability during training and perhaps improve performance. As there are more layers here than in the first CNN model, the model will be able to learn more intricate associations between the flattened features and the output. The flatten () layer is also used to convert to 1D for fully connected layer, where 256 neurons are employed, an increase from the first CNN model. Additionally, during training, the Dropout () layer eliminates 50% of the activations from the preceding layer at random

to prevent overfitting. The output is then normalised. Finally, a completely linked layer with two neurons employs a sigmoid distribution to give a probability distribution result over the possibility of identifying pneumonia.

Then, same as model 1, the optimiser Adam and decreasing learning approach are applied here as well.

Same procedure as previous model is used for training with callbacks.

4.2.3 Model 3

Model 03 - CNN model from scratch with Five Conv and 2 FC Layers

```
# Define input shape
input_shape = (224, 224, 3)
# Define the model
CNN_model = tf.keras.Sequential([
    # Convolutional layer 1
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),
    # Convolutional layer 2
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),
    # Convolutional layer 3
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),
    # Convolutional layer 4
    layers.Conv2D(256, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),
    # Convolutional layer 5
    layers.Conv2D(512, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),
    # Flatten the output from convolutional layers
    layers.Flatten(),
    # Fully connected layer 1
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.5),
    layers.BatchNormalization(),
    # Fully connected layer 2
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.BatchNormalization(),
    # Output layer
    layers.Dense(2, activation='sigmoid')
])
```

This model contains five convolutional layers, with the first layer looking for patterns in the picture and the fifth layer recognising complicated characteristics from the image using 512 neurons. Like the previous model, this one also uses relu to ensure linearity, followed by max pooling to reduce special dimensions, and normalisation in each layer. The output for the completely linked layer is then flattened and converted to 1D. As a result of having more convolutional layers than previous models, this model contains two fully connected layers, which will help it learn more intricate and detailed representations of the input picture and enhance the model's performance. To prevent

overfitting and to achieve better results, dropout and normalisation are also utilised on those layers. To create a probability distribution of it, the output of the last fully connected layer is fed into the output layer using a sigmoid function.

Same procedure as previous model is used for training with callbacks.

4.2.4 ResNet50 Pre-trained model

Model: Pre-trained - ResNet50

```
[ ]  #Downloading Resnet50 model
     resnet_model = ResNet50( weights = 'imagenet',include_top = False, input_shape = (image_size,image_size,3))
     resnet_model.summary()
```

```
     #printing layers of the pretrained ResNet50 model
     for i, layer in enumerate(resnet_model.layers):
         print(i, layer.name)
```

```
     #adding layers at the top of the classifier
     model = resnet_model.output
     model = tf.keras.layers.GlobalAveragePooling2D()(model)
     model = tf.keras.layers.Dropout(rate=0.2)(model)
     model = tf.keras.layers.Dense(2,activation='sigmoid')(model)
     model = tf.keras.models.Model(inputs=resnet_model.input, outputs = model)
```

The pre-trained ResNet50 model is produced using the Keras ResNet50 tool. Using pre-trained weights or starting from scratch when training the model is determined by the weights parameter. The imagenet dataset, which contains more than one million tagged photos, is used to initialise the model in this case, improving the model's performance and accuracy for image detection. Include_top Specifies whether to include the fully linked layer at the top of the model. If you set it to False, you may create a custom top layer for specific task in this example, the identification of pneumonia. The model's input data, which is 224x224 and has three RGB colour channels, is represented by the shape input_shape. To develop a new model that would be trained exclusively for pneumonia identification, layers are added on top of the pre-trained ResNet50. The ResNet50 model's output is extracted via the function resnet_model.output. The model's next line adds a global average pooling layer that averages each feature map in the output of the layer before that to create a single value for each feature map. The output's dimensionality is decreased by reducing the number of model parameters, which also aids in preventing overfitting. The model's capacity is then reduced by a random 20% activation drop, which forces the model to discover more robust features and avoids overfitting. A fully connected dense layer with two neurons and a sigmoid is added to the model in the line that follows, producing a probability distribution across the two potential classes, "PNEUMONIA" or "NORMAL," which represent the presence or absence of a pneumonia. In the final line, a new Keras model is created using the ResNet50 model as the input layer and the new layer that was placed on top of the ResNet50 model as the output layer. This allows the model to be reused or customised as necessary.

Finally, the previously described process was repeated to optimise, save the best model, lower the learning rate, and train the model.

4.2.5 EfficientNetB1 Pre-trained model

Model: Pre-trained - EfficientNetB1

```
[ ]  1  #load pre-trained EfficientNet weights
     2  effnet = EfficientNetB1(weights='imagenet',include_top=False,input_shape=(image_size,image_size,3))
```

```
[ ]  1  #adding custom layers at the top of the classifier
     2  model = effnet.output
     3  model = tf.keras.layers.GlobalAveragePooling2D()(model)
     4  model = tf.keras.layers.Dropout(rate=0.5)(model)
     5  model = tf.keras.layers.Dense(2,activation='sigmoid')(model)
     6  model = tf.keras.models.Model(inputs=effnet.input, outputs = model)
```

EfficientNet utilises the same imagenet dataset as the previous pre-built model and then adds custom layers on top.

The additional layers are the same as the additional layers of ResNet50; they use the output layer from the original EfficientNet and include global average pooling and dropout to avoid overfitting. The probability distribution of the "PNEUMONIA" and "NORMAL" classes also includes a sigmoid. The output layer is additional layers, and the last line is the initial EfficientNet input layer.

Optimising, saving the best model, lower the learning rate, and training the model is also done here.

4.2.6 VGG-16 pre-trained model

```
[ ]  1  # Defining and Laoding the pretrained VGG16 model
     2  np.random.seed(42)
     3  base_model = VGG16( weights='imagenet',include_top=False, input_shape=(224,224,3))
```

```
[ ]  1  #printing layers of the model
     2  for i, layer in enumerate(base_model.layers):
     3      print(i, layer.name)
```

```
[ ]  1  for layer in base_model.layers:
     2      layer.trainable = False
```

```
[ ]  1  # finetunning - freezing early 8 layers and training later layers of the model
     2  for layer in base_model.layers[:8]:
     3      layer.trainable = False
     4  for layer in base_model.layers[8:]:
     5      layer.trainable = True
```

```
[ ]  1  # adding custom layers at the top of the pretrained classifier
     2  NUM_CLASSES = 2
     3  model2 = Sequential()
     4  model2.add(base_model)
     5  model2.add(Flatten())
     6  model2.add(Dense(1024, activation = 'relu'))
     7  model2.add(Dropout(0.5))
     8  model2.add(Dense(NUM_CLASSES, activation='sigmoid'))
```

```
[ ]  1  # compiling the model
     2  model2.compile(loss='binary_crossentropy',optimizer=Adam(lr=1e-4),metrics=['accuracy'])
     3  model2.summary()
```

```
[ ]  1  callback = EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=0,mode='auto', baseline=None, restore_best_weights=False)
```

Training of the model

```
epochs = 50
history = model2.fit(X_train2, y_train2, validation_split = 0.1 , epochs = epochs, verbose = 1, batch_size = 32, callbacks = callback)
```

The random seed is set to 42, which ensures a consistent outcome every time the code is executed. Then, the vgg16 base model employs the same parameters as earlier pre-trained models. The pre-trained layers of the vgg16 are frozen at layer.trainable = False. The first eight layers of the Vgg16 model are frozen, while the bottom eight levels are utilised to extract information from the input pictures before fine-tuning the top layer to identify pneumonia. Using a portion of the pre-trained model for fine-tuning expedites training and prevents overfitting.

On top of that, custom layers have been introduced. The output layer's num_classes value is 2, indicating a binary categorization of either "PNEUMONIA" or "NORMAL,". Keras uses sequential () to construct neural networks. The performance is then improved by adding a pre-trained model to a new sequential model, which will inherit its weights. To produce 1D, a flatten layer is applied before being sent to the completely linked layer. Then, 1024 with the relu function is used in the fully linked layer to make it non-linear. The model will be able to understand more intricate characteristics and patterns in the photos because to the increased number of neurons. Additionally, there is a chance dropout of 50% activation, which restricts the model's potential. The previously described process was repeated to optimise, save the best model, lower the learning rate, and train the model.

Finally, the previously described steps for optimising, saving the best model, lowering the learning rate, and training the model were repeated. In this model, the callback mechanism is modified to early stopping, which reduces overfitting and boosts training effectiveness. Additionally, it was stated that training would end if validation loss did not improve after 10 iterations, which was a patient measure to avoid overfitting.

4.2.7 DenseNet121 Pre-trained model

Model: Fine Tuned Pre-trained - DenseNet121

```
#Defining and Loading pretrained DenseNet121 model
densenet_model = DenseNet121(weights='imagenet',include_top=False,input_shape=(224,224,3))
```

```
#Printing all layers of the pretrained model
for i, layer in enumerate(densenet_model.layers):
    print(i, layer.name)
```

```
for layer in densenet_model.layers:
    layer.trainable = False
```

```
# finetunning - freezing early 141 layers and training later layers of the model
for layer in densenet_model.layers[:141]:
    layer.trainable = False
for layer in densenet_model.layers[141:]:
    layer.trainable = True
```

```
#adding custom layers at the top of the model
NUM_CLASSES = 2
model = Sequential()
model.add(densenet_model)
model.add(GlobalAveragePooling2D())
model.add(Flatten())
model.add(Dense(512,activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(NUM_CLASSES, activation='sigmoid'))
```

```
# Compiling the model
model.compile(loss='binary_crossentropy',optimizer=Adam(lr=1e-3),metrics=['accuracy'])
model.summary()
```

Training of the Model

```
epochs = 50
history = model.fit(X_train2, y_train2, validation_split = 0.1 , batch_size = 16, epochs = epochs, verbose = 1)
```

DenseNet additionally incorporates layers from prior pre-trained models on top of the ImageNet dataset. The first 141 layers of the pre-trained DenseNet model are frozen, while the remaining 141 layers are adjusted for training. This enhances speed, enables feature extraction from input photos, and avoids overfitting. Then, a custom layer with pre-built DenseNet, global average pooling, flatten layer, and 512 neurons with relu was added to the fully connected layer to learn intricate features and patterns from pictures. Finally, a sigmoid distribution is employed to determine if a pneumonia is present or absent. The model was then trained using optimiser, learning similarly to prior models.

5 Experimental Results

In addition to recall, precision, f1-score, and ROC curve as assessment measures, accuracy curves and loss curve graphs will also be utilised to assess how well the CNN models performed. To evaluate the effectiveness of deep learning models across many domains, several measures are utilised. To determine if a model is overfitting or underfitting, accuracy curves and loss curves graphs show how the accuracy and loss of the model vary throughout training. Knowing the model's true positive and true negative ratesâwhere it properly identified positive classes like pneumonia and correctly predicted negative classes like ânormalâ is essential because this is a classification problem. Recall, accuracy, f1-score, and ROC are used to evaluate the model's effectiveness and pinpoint areas for improvement. Recall is the model's capacity to accurately recognise all positive samples. The fraction of accurate positive predictions, or precision, is measured. F1-score offers a score that strikes a balance between recall and accuracy. The model's ability to differentiate between positive and negative samples is represented by the ROC curve. The following equations are given for them:

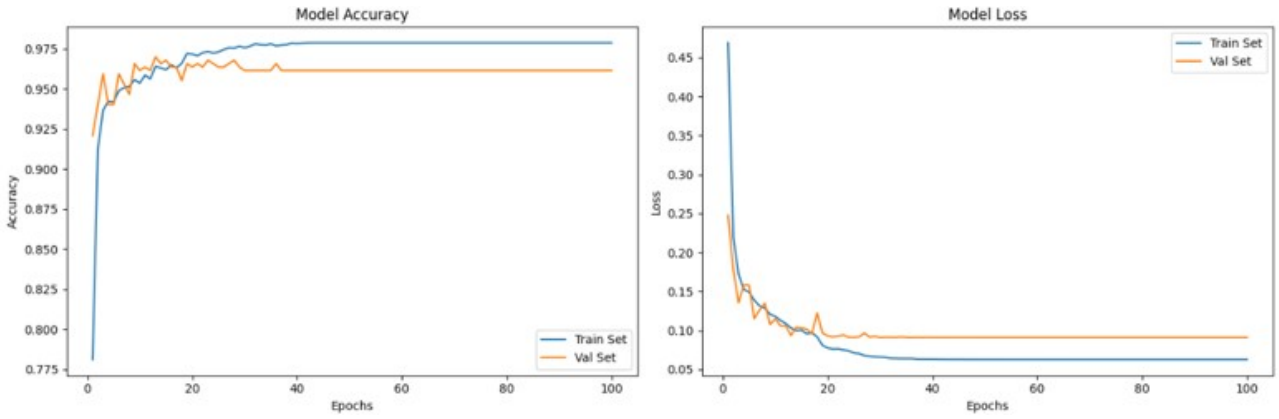
$$
\begin{aligned}
precision &= \frac{TP}{TP + FP} \\
recall &= \frac{TP}{TP + FN} \\
F1 &= \frac{2 \times precision \times recall}{precision + recall} \\
accuracy &= \frac{TP + TN}{TP + FN + TN + FP}
\end{aligned}
$$

Results of Trained model on Without Augmentation Dataset				
Model	**Test Accuracy**	**Precision**	**Recall**	**F1-Score**
CNN model (3 layers)	94.2	0.9180	0.8765	0.8968
CNN model (3 layers balanced classes)	98.56	0.9873	0.9852	0.9863
CNN model (4 layers)	98.95	0.9884	0.9915	0.9900
CNN model (5 layers)	99.39	0. 9947	0. 9937	0. 9942
Pre-trained ResNet50	99.89	1.0000	0. 0.9979	0. 0.9989
Pre-trained EfficientNetB1	99.61	0. 9968	0. 9958	0. 9963
Pre-trained Fine Tunned VGG16	98.51	0. 0.9852	0. 0.9863	0. 0.9857
Pre-trained Fine Tuned DenseNet121	99.34	0. 9905	0. 9968	0. 9937

Here, from the table above it shows after performing data augmentation and making the classes balanced the model improved, therefore for all the model after it used the augmented dataset. Models from scratch have been gradually improving with model 3 outperforming all. From the pre-trained model ResNet50 done the best followed by EfficientNet, vgg16 and DenseNet121.

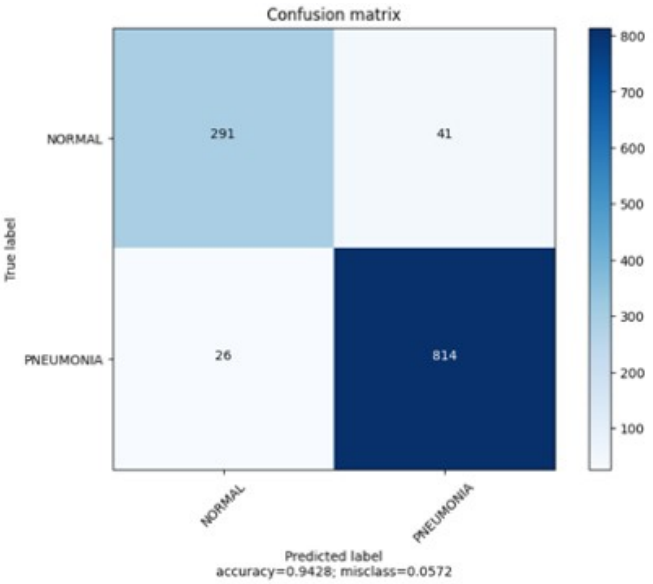
Model 1

Three convolutional layers, including two fully linked layers, were added to the original model. In addition to the layers indicated in the implementation section, the max pooling layer, relu activation, and flatten layer have also been used. Additionally, it makes use of the Adam optimizer, the loss function, and lowers the learning rate if the loss function is unchanged for five epochs.



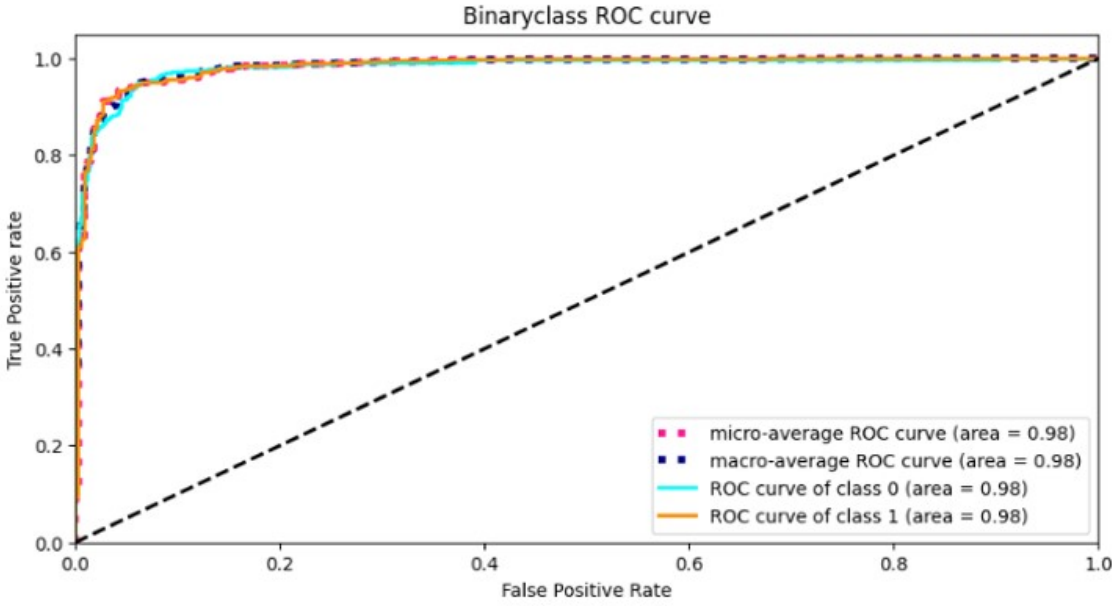
The loss curve demonstrates that the loss between the models differs but not much. Given the large gap between the two classes and it is possible that the training dataset lacked enough data to perform effectively on the validation set. Validation loss is higher than training loss meaning it needs improving for learning rate.

	precision	recall	f1-score	support
NORMAL	0.9180	0.8765	0.8968	332
PNEUMONIA	0.9520	0.9690	0.9605	840
accuracy			0.9428	1172
macro avg	0.9350	0.9228	0.9286	1172
weighted avg	0.9424	0.9428	0.9424	1172



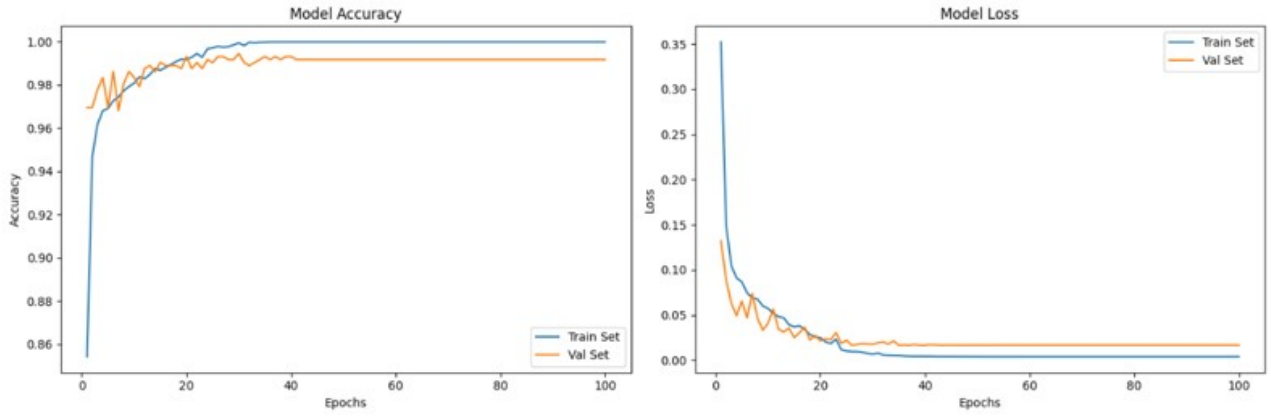
The accuracy, precision, recall, and f1-score all have decent results in the classification report above, all of which are over 85%. This indicates that the model's positive predictions are fairly accurate and that it was successful in accurately identifying the actual positive events. According to the confusion metrics, the model accurately predicted 814 Pneumonia images out of 840 actual pneumonia images, which means it misidentified 26 images as normal while they were pneumonia. Due to the model's inaccurate identification, the patients may not receive treatment, which might be fatal. Later models might make improvements to this. Out of 332 actual normal images, the model correctly identified

291 as normal, but 41 of those were incorrectly identified as pneumonia, which can be expensive for hospitals because it necessitates making unneeded diagnoses. This will be improved in later models.



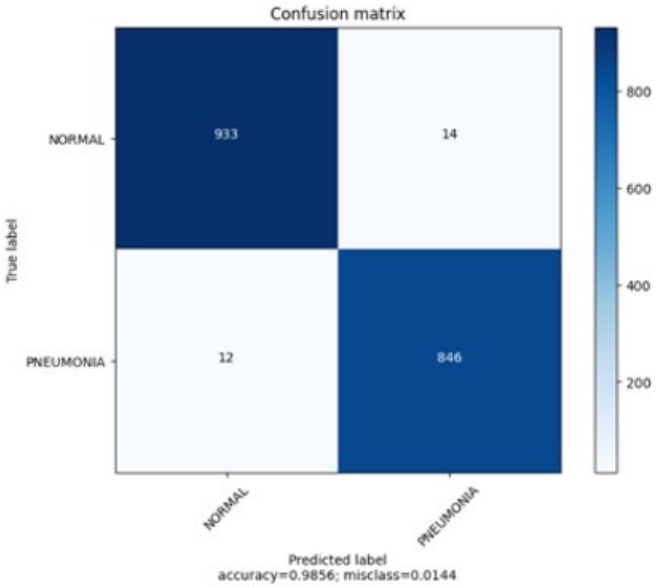
The ROC curve graph shows how well a model can differentiate between two states, such as "Pneumonia" or "Normal,". It illustrates how frequently the model is accurate known as the true positive rate when it responds with " Pneumonia " or " Normal,â and how frequently it is incorrect known as the false positive rate when it responds with the output " Pneumonia." The curve is created by altering the decision threshold that the model employs and visualising the outcomes. When the curve is higher, the model is accurate at noticing difference between two things. The ideal curve, which represents flawless categorization, is the one that pierces the upper left corner of the plot. Curve is high in this graph, indicating that there are more false positives and that the model is effective at detecting pneumonia.

Model 1 with balanced dataset:

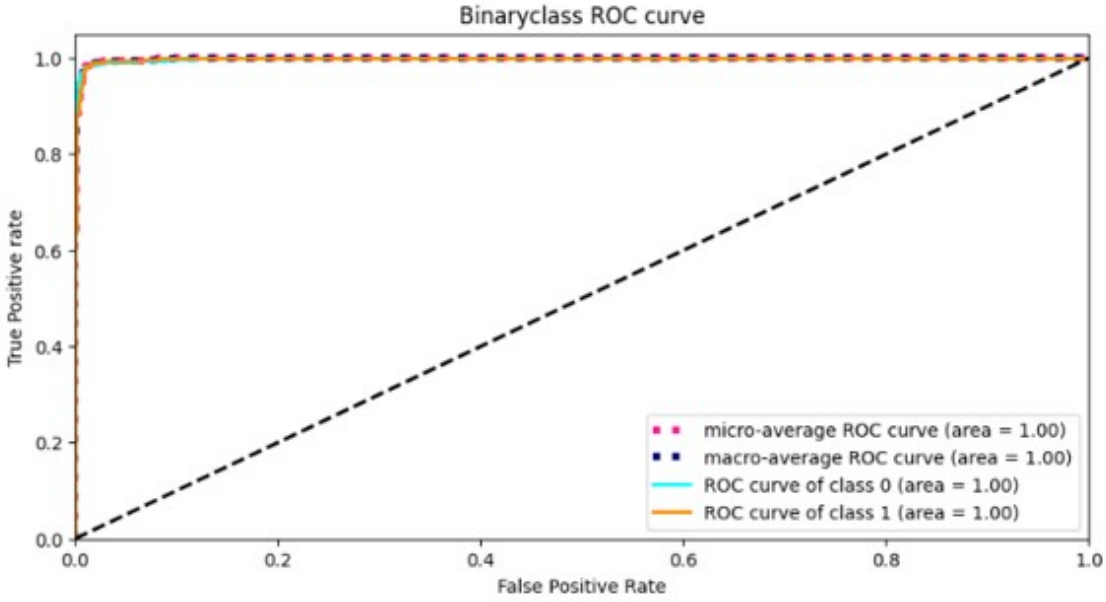


Graph shows that the validation loss is higher than training loss after 20 epochs mean modelâs generalising ability goes down which needs to be improved. Early stopping can be introduced, but the model seems to be improving as the loss went down over time.

	precision	recall	f1-score	support
NORMAL	0.9873	0.9852	0.9863	947
PNEUMONIA	0.9837	0.9860	0.9849	858
accuracy			0.9856	1805
macro avg	0.9855	0.9856	0.9856	1805
weighted avg	0.9856	0.9856	0.9856	1805



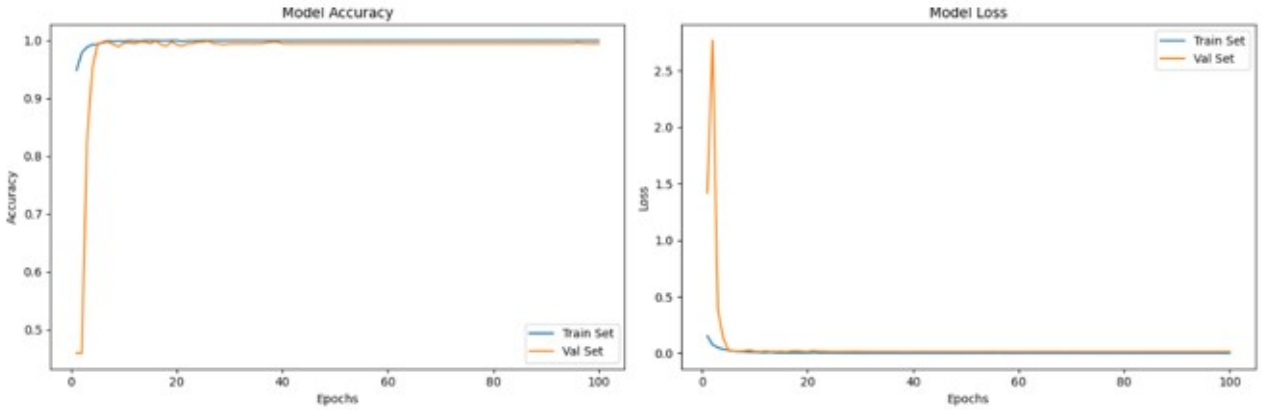
Here, accuracy, precision, recall, and f1-score have increased significantly this is due to adding more augmented images to âNORMALâ class and making the dataset balanced. According to the confusion metrics, the model accurately predicted 846 Pneumonia images out of 858 actual pneumonia images, which means it misidentified 12 images as normal while they were pneumonia. This has improved. Later models might make improvements to this. Out of 947 actual normal images, the model correctly identified 933 as normal, but 14 of those were incorrectly identified as pneumonia.



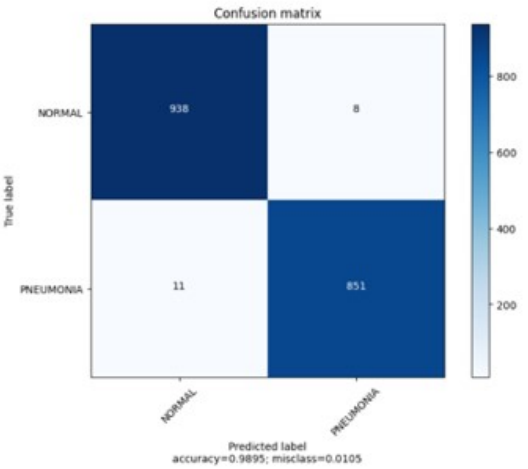
Since the ROC curve in this graph is 1, which indicates that it has hit its maximum and can properly detect most pneumonia from the dataset, the outcome is excellent. This can mean that the model is functioning well.

Model 2

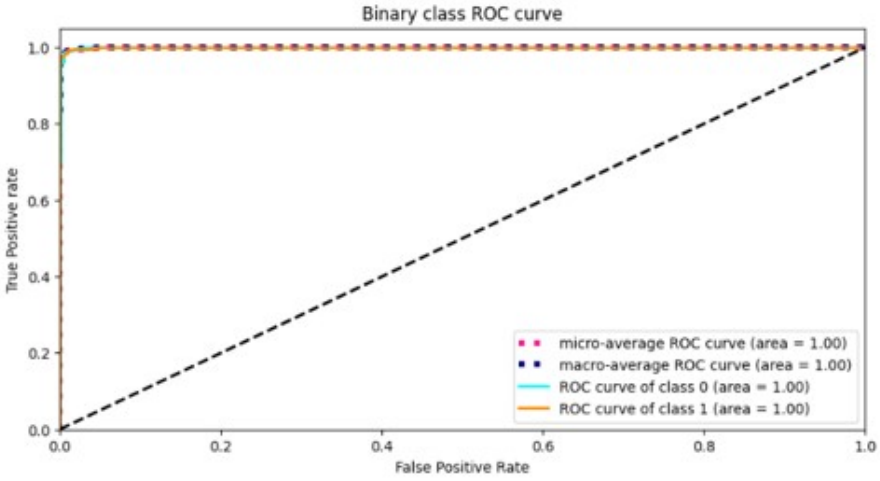
Good results are shown by the model training and validation curve. The validation accuracy wasn't performing well at first, but after a few epochs, it became more accurate and lost less data. The strategy that reduces learning rate if validation doesn't drop and increases model generalisation is again to blame for this. As accuracy rose training and validation loss also dropped.



	precision	recall	f1-score	support
NORMAL	0.9884	0.9915	0.9900	946
PNEUMONIA	0.9907	0.9872	0.9890	862
accuracy			0.9895	1808
macro avg	0.9895	0.9894	0.9895	1808
weighted avg	0.9895	0.9895	0.9895	1808



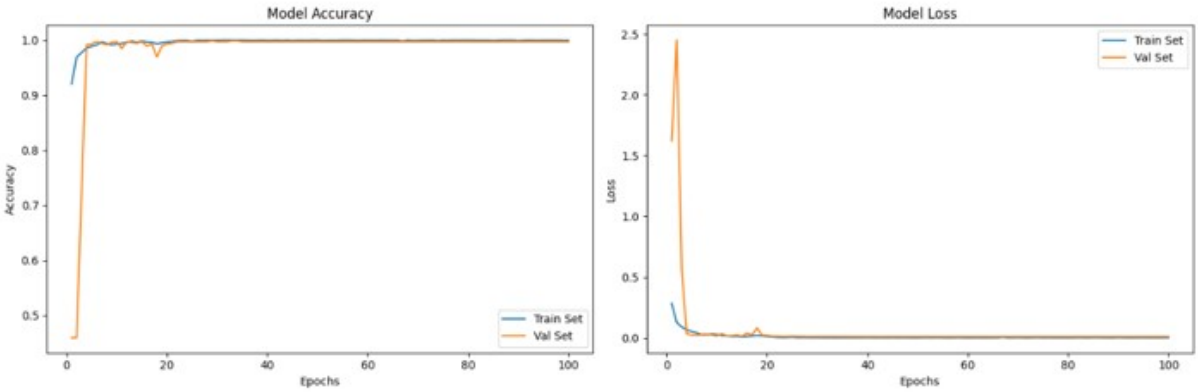
This model shows good accuracy, precision, recall, f1-score, and this has improved from previous model. As it incorrectly predicted 8 Normal and 11 pneumonias from 946, 862 respectively. Meaning this will be beneficial for patient and cost effective for the hospitals. This ROC curve also got 1,



which means that it has hit its maximum and can properly detect most pneumonia from the dataset, the outcome is excellent. This can mean that the model is functioning well.

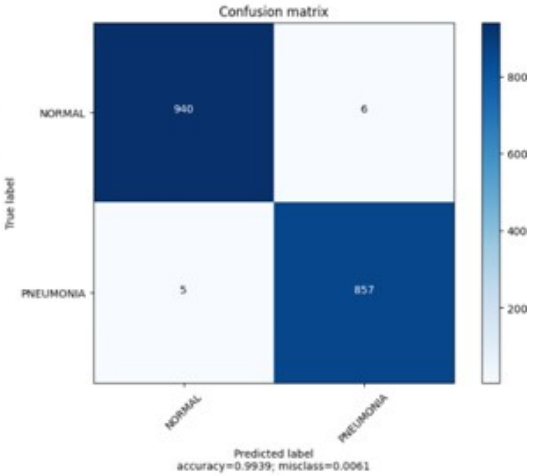
Model 3

The performance of the training and validation curves improves as more data is provided. The accuracy initially wasn't very excellent, but after a few epochs it started to improve. Later, it

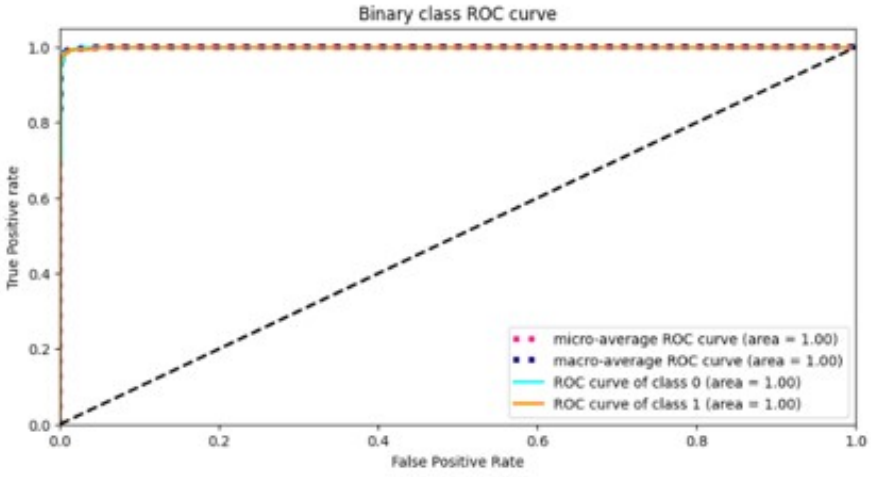


fluctuated a bit because of the model's addition of optimisation. As additional epochs were completed, training loss likewise significantly decreased along with validation loss.

	precision	recall	f1-score	support
NORMAL	0.9947	0.9937	0.9942	946
PNEUMONIA	0.9930	0.9942	0.9936	862
accuracy			0.9939	1808
macro avg	0.9939	0.9939	0.9939	1808
weighted avg	0.9939	0.9939	0.9939	1808



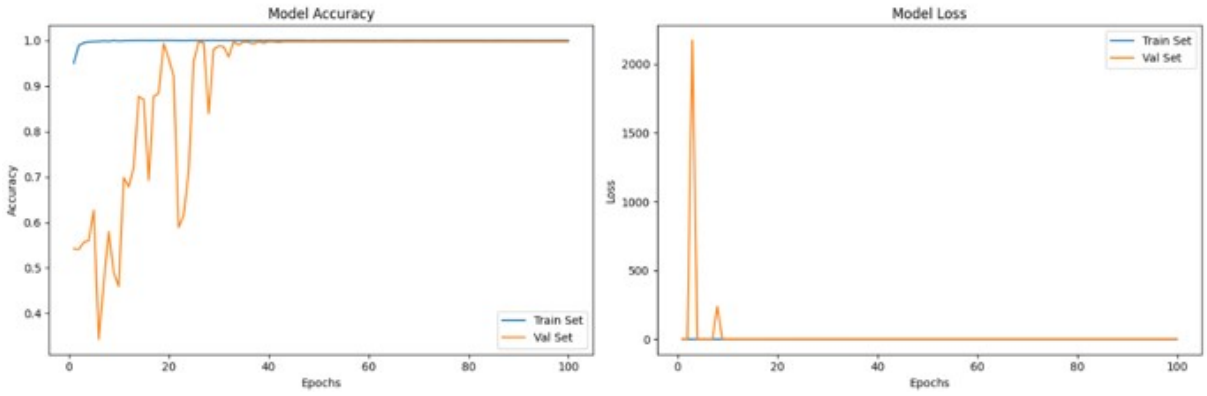
This model shows excellent improved accuracy, precision, recall, f1-score. As it incorrectly predicted only 6 Normal and 5 pneumonias from 946, 862 respectively. Meaning this model is reliable to be used for patient as well as its effective for the hospitals as it will save money, because they will not have to do unnecessary check ups and waste resources.



This ROC also suggests that the model is performing well and reliable.

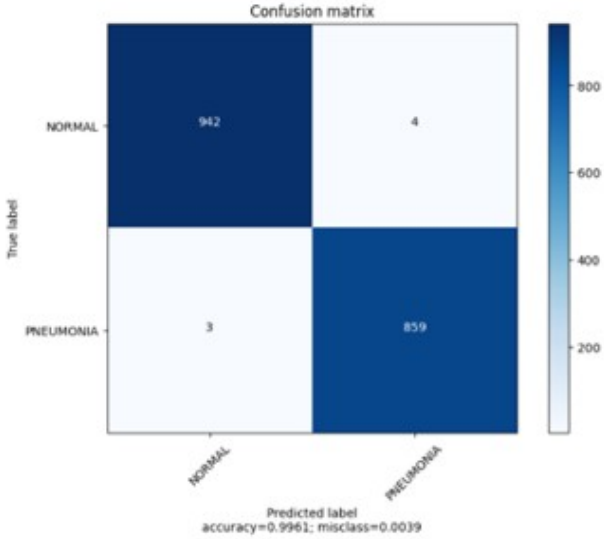
Pre-trained model: EfficientNetB1

All these evaluation procedures have been done for all the pre-trained model. Here only Efficient-NetB1will be shown as they are quite similar.

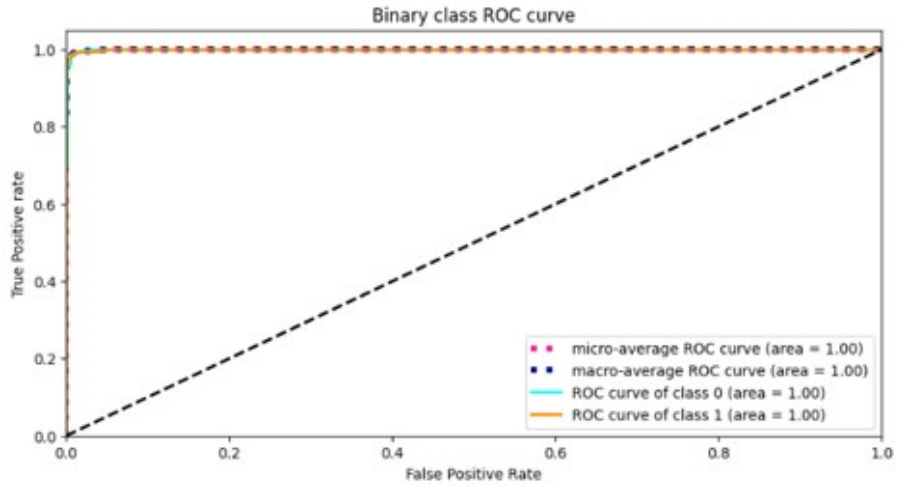


The training and validation curves demonstrate that while the validation accuracy initially struggled and fluctuated over time, it began to improve after 40 epochs. This is because EfficientNetB1 requires a substantial quantity of training data, which we lacked. As a result, by including the dropout at the top of the model, it was possible to overcome the validation loss and gradually increase validation accuracy

	precision	recall	f1-score	support
NORMAL	0.9968	0.9958	0.9963	946
PNEUMONIA	0.9954	0.9965	0.9959	862
accuracy			0.9961	1808
macro avg	0.9961	0.9961	0.9961	1808
weighted avg	0.9961	0.9961	0.9961	1808



Result here also shows model have performed very well. As the accuracy, precision, f1-score, recall is all above 99%. Only 4 was incorrectly predicted as normal and 3 was incorrectly predicted as pneumonia.



This Roc curve is also showing how reliable and excellent the model has performed.

6 Application Phase

Application Phase

```
#Loading Un-seen Image For images
#img_path = '/content/drive/MyDrive/DNN/Chest_X-ray/Data_APP_Phase/PNEUMONIA.jpg'
# For NORMAL
img_path = '/content/drive/MyDrive//DNN/Chest_X-ray/Data_APP_Phase/NORMAL.jpg'
```

```
#Loading and Pre-Processing Image
image_size = 224
image = cv2.imread(img_path)
#image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
image = cv2.resize(image,(image_size, image_size))
image_array = np.array(image)
```

Load the trained model

```
#Load the best saved trained model
best_model = load_model('/content/drive/MyDrive/DNN/Chest_X-ray/Implementation/Saved_Models/ResNet50.h5')
```

Prediction on Unseen Instance

```
# Making prediction on Unseen Image using Trained model
image_array = np.expand_dims(image_array, axis=0)
prediction = best_model.predict(image_array)
prediction = np.argmax(prediction,axis=1)
print(prediction)
```

```
1/1 [==============================] - 0s 38ms/step
[0]
```

The code to import an image file into the model for processing is shown in the above picture. There are two pathways; one loads an Images of a NORMAL but the other is commented out. The application's second path loads Pneumonia. The input picture is then loaded and pre-processed so that the model can do prediction on it.

The best model is then loaded from the file location for usage in the application phase. The best model in this instance was ResNet50. The algorithm then uses a pretrained model to forecast once the picture has been loaded and pre-processed. The first step in creating the prediction is to

increase the 'image_array' variable's size to correspond to the model's predicted input form. This is accomplished by utilising the 'np. expand_dims' method, which begins the array with an additional dimension.

When generating predictions on a single image, the batch size of the input data is commonly set to 1, which corresponds to this dimension. Next, the enlarged 'image_array' is passed as input to the 'predict' function, which is run on the 'best_model' variable. This function creates a forecast for the input picture using the pre-trained model. A numpy array containing predicted probabilities for each class in the model's output is the result of the 'predict' function.

The 'argmax' function is then used by the algorithm to locate the class index with the highest predicted probability. The anticipated class label for the input picture is represented by this index. Finally, the 'print' function is then used to print the expected class label to the terminal.

Prediction on Unseen Instance

```
# Making prediction on Unseen Image using Trained model
image_array = np.expand_dims(image_array, axis=0)
prediction = best_model.predict(image_array)
prediction = np.argmax(prediction,axis=1)
print(prediction)
```

```
1/1 [==============================] - 2s 2s/step
[1]
```

```
#Ploting the predictions
if prediction[0] == 0:
  print("\033[1m","\n\nTrained Model Prediction")
  print("\033[1m","+","="*30,"+")
  print("\033[1m","|"," "*30,"|\n           Class : Normal Image         \n","    |                              |")
  print("\033[1m","+","="*30,"+")
  plt.imshow(image)
  plt.xticks([]), plt.yticks([])  # to hide tick values on X and Y axis
  plt.show()

else:
  print("\033[1m","\n\nTrained Model Prediction")
  print("\033[1m","+","="*30,"+")
  print("\033[1m","|"," "*30,"|\n           Class : Pneumonia Image       \n","|                              |")
  print("\033[1m","+","="*30,"+")
  plt.imshow(image)
  plt.xticks([]), plt.yticks([])  # to hide tick values on X and Y axis
  plt.show()
```

```
Trained Model Prediction
+ ============================== +
|                                |
          Class : Normal Image
|                                |
+ ============================== +
```

The above code is used to plot predictions provided by a trained model on an input picture. The code outputs a message to the console informing the user that the model correctly predicted a normal image if the predicted class label is â0â Then, using the 'plt.imshow' and 'plt.xticks' and 'plt.yticks' methods, it displays the input picture and conceals the tick values on the x and y axes, respectively.

The code outputs a message to the console informing the user that the model anticipated a pneumonia image if the predicted class label is not "0", which corresponds to a pneumonia image. Then, using the 'plt.imshow' and 'plt.xticks' and 'plt.yticks' methods, it displays the input picture and conceals the tick values on the x and y axes, respectively.

7 Summary

In this report, we examined how well several convolutional neural network models performed the job of detecting pneumonia. To improve the performance of the models, batch normalisation and other optimisations were implemented, and each model had an increasing number of layers and layers. The results revealed that the models created from scratch performed better in the second and third model because they had more layers, more neurons, and other performance-enhancing factors. They also had a higher accuracy rate. ResNet50 achieved the greatest accuracy rate of 99.89% among the pre-built models, which in general outperformed the custom-built models. Given their sophisticated design and training methods like transfer learning, pre-built models are quite successful. They have also been taught on a vast quantity of data, which normally improves their performance because they are better learners.

However, properly constructed and trained custom built models that were developed with an adequate number of layers, neurons, and learning rates to prevent overfitting or underfitting in the model can also obtained great accuracy, this report is a proof of that.

8 Future Work

The results of this study have led to several recommendations for future research that might expand on current work and enhance the efficiency of convolutional neural network models for the classification of pneumonia.

Investigating the use of segmentation algorithms to separate the pneumonia region in X-ray images is one possible area of future research. By lessening the influence of pneumonia infected areas on the categorization decision, this can increase the accuracy. For treatment planning and monitoring, segmentation can also assist to offer more precise information on the location of pneumonia.

9 References

- Shortliffe, E.H. and Blois, M.S., 2006. The computer meets medicine and biology: emergence of a discipline. Biomedical informatics: Computer applications in health care and biomedicine, pp.3-45.

- Verma, D., Bose, C., Tufchi, N., Pant, K., Tripathi, V. and Thapliyal, A., 2020. An efficient framework for identification of Tuberculosis and Pneumonia in chest X-ray images using Neural Network. Procedia Computer Science, 171, pp.217-224.

- E. Ayan and H. M. Ãnver, "Diagnosis of Pneumonia from Chest X-Ray Images Using Deep Learning," 2019 Scientific Meeting on Electrical-Electronics Biomedical Engineering and Computer Science (EBBT), Istanbul, Turkey, 2019, pp. 1-5, doi: 10.1109/EBBT.2019.8741582.

- Elshennawy, N.M. and Ibrahim, D.M., 2020. Deep-pneumonia framework using deep learning models based on chest X-ray images. Diagnostics, 10(9), p.649.

- Kundu, R., Das, R., Geem, Z.W., Han, G.T. and Sarkar, R., 2021. Pneumonia detection in chest X-ray images using an ensemble of deep learning models. PloS one, 16(9), p.e0256630.

- Ibrahim, A.U., Ozsoz, M., Serte, S., Al-Turjman, F. and Yakoi, P.S., 2021. Pneumonia classification using deep learning from chest X-ray images during COVID-19. Cognitive Computation, pp.1-13.