Pattern Recognition

Project Two

Chest X-Ray Diagnosis

Presented by:

Ayman Ahmed	@aymanelghotni
Mostafa Hakam	@MostafaHakam98

CONTENTS

Summary

Problem Statement:

It was required to make a Chest X-Ray classifier that distinguishes between COVID-19 infected lungs, Viral Pneumonia and healthy lungs using several deep learning architectures and compare their efficiency to the mentioned task.

Solution Approach:

The solution approach was done in three steps. The first step pre-processes the data and creates data pipelines, then explores several deep learning architectures beginning with a simple shallow fully connected neural network and another deep fully connected neural network then passing by a shallow convolutional neural network and a deep convolutional neural network and finally ending with famous convolutional neural networks such as: ResNet50, DenseNet121 and InceptionV3.

The last step is to apply gradient class activation mappings (GRAD-CAM) to segment the areas that mostly affected the decision of the model. GRAD-CAM is only applied to convolutional neural networks.

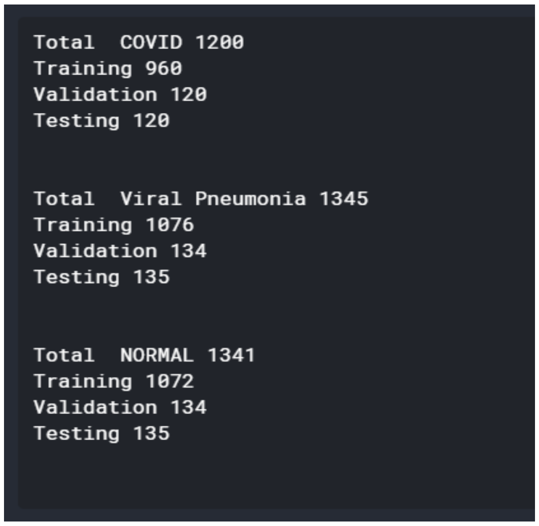
Libraries used:

1. Library “numpy”.
2. Library “pandas”.
3. Library “tensorflow”.
4. Library “matplotlib”.
5. Library “keras”.
6. Library “os”.
7. Library “sklearn”.
8. Library “PIL”.

Data Processing

About Data:

Data is imported from Kaggle Library which a team of researchers have provided. The database contains chest X-ray images for COVID-19 positive cases along with Viral Pneumonia and normal cases images. It consists of 1200 positive COVID-19 images, 1345 viral pneumonia images, and 1341 normal images, as illustrated in figure (1).



```
Total  COVID 1200
Training 960
Validation 120
Testing 120


Total  Viral Pneumonia 1345
Training 1076
Validation 134
Testing 135


Total  NORMAL 1341
Training 1072
Validation 134
Testing 135
```

Figure 1

Splitting:

Data are firstly loaded from the database (COVID-19 Radiography Database) into the session then are splitted randomly with a constant seed through a for-loop into three folder: Training, Validation, and Testing, which are splitted as 80%, 10%, and 10% respectively, as shown in figure (2). Each folder consists of three sub-folders; Covid-19, Pneumonia, and Normal.

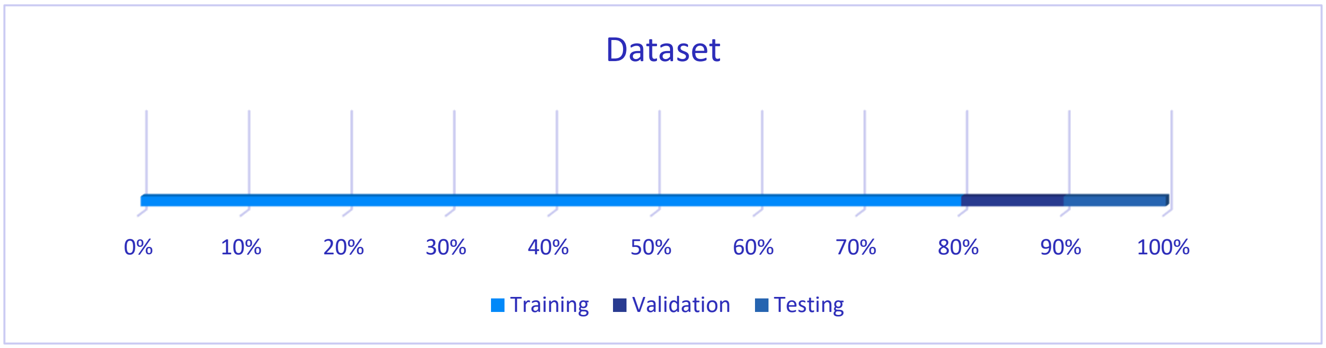


Figure 2

Data loading:

Data is loaded by ImageDataGenerator class which applies random real time augmentation if provided. The augmentation used was horizontal flipping of the image, rotation range of 10 degrees, and zoom range of 0.5. All the mentioned augmentations are randomly applied to prevent overfitting of models and to provide more data to training with no need for more images. The augmentation is done only on the training set as augmenting the test and validation data has no use because data is used to evaluate the performance of the model.

It was found in the exploration of the data that they have different sizes so the images are loaded to a fixed target size of 256×256×3 as RGB images. Although the images in the dataset are in grayscale, but it was required to have 3 channels for the feeding of the images inside the networks. So, by loading the images in RGB format it is loaded as 3 grayscale images concatenated on the channels axis.

The different images sizes are either reduced if they were larger than the target size or increased if they were smaller than the target size, the reducing/increasing algorithm is done using the bicubic interpolation algorithm which is an extension of cubic interpolation for interpolating data points on a two-dimensional regular grid, as given in the equation in figure (4). The interpolated surface is smoother than corresponding surfaces obtained by bilinear interpolation or nearest-neighbor interpolation, as shown in figure (3).

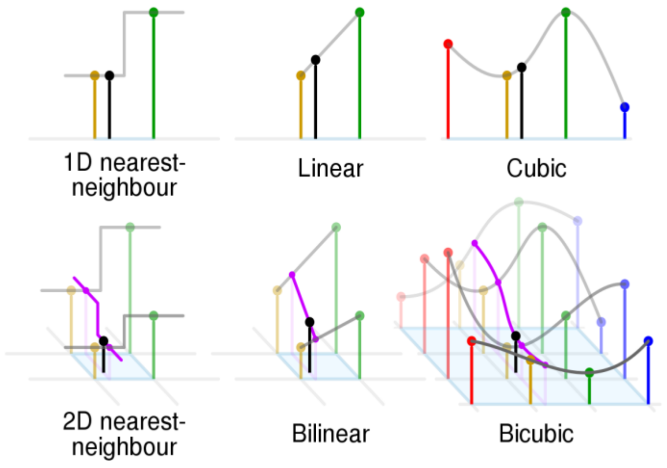


Figure 3

The data loaded for training and validation are shuffeled each time, as the shuffeled data are consistent to the stratification of classes so that no batch has a dominant class, but the data loaded for testing isn't because it isn't necessary because it is used for evaluating the model, and the order doesn't matter.

$$p(x,y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j .$$

Figure 4

Data Processing

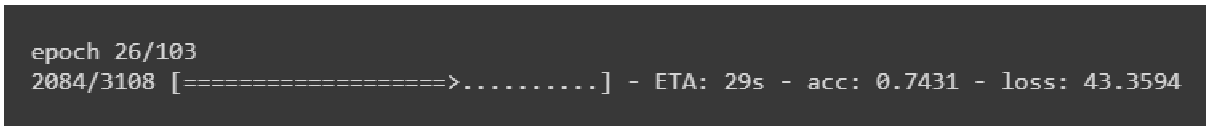
Data preprocessing:

Before any image enter the neural networks for training or evaluation, it is standarized first, the standarization is done by calling the Normalization layer from keras and is adapted on the training set only to remain unbiased and because the model shouldn't have any intuition about how the test set looks like or any information about it, the standarization is done on pixel level as every pixel is one dimension so we get the mean and standard deviation of each position of the pixel and then then every pixel in the input image is standarized so that the total image has a mean of zero and standard deviation equals 1.

Training & Evaluation

Training and Validation:

A training function is defined to take a specific optimizer, its metric values, for training and validation, and the number of epochs to train the given model. It consists of one loop, in which an iteration represents an epoch through a progress bar, as shown in figure (5), containg two nested loops, one for training and another for validation. Each nested loop consists of a number of procedures, first is to normalize the given batch of data then calculate the accuracy and the total loss.



```
epoch 26/103
2084/3108 [===================>..........] - ETA: 29s - acc: 0.7431 - loss: 43.3594
```

Figure 5

Each loop, either training or validation, continuiously call a corresponding one step function. The training-one-step function applies gradients to the optimzer of the model then calculates the loss of the ingoing batch and returns it. The validation-one-step function does a similar job except that it doesn't deal with the optimizer or gradients in any way.

Callbacks are implemented to provide consistency to the model. Early stop is provided in form of an int (patience) which defines the number of epochs in which the model is evaluated on a holdout validation dataset after each epoch. If the performance of the model on the validation dataset finds no further progress, then the training process is stopped. The patience in this case is 10. Another callback is the learning rate scheduler. A learning rate scheduler seeks to reduce the learning rate during training according to a schedule using inverse-time-decay algorithm.

The training function finally set the weights to the model, which found the lowest loss for the validation set, and returns two arrays of losses and accuracies for both training and validation set.

Evaluation:

An evaluation function is provided for the test set. The architecture of the function is similar to those of the training and the validation's. It consists of a for -loop that normalizes the data and computes the loss for every batch. The evaluation function returns the confusion matrix of the test set and a report containing the accuracy, recall, precision, and f1-score.

Optimizer:

The optimizer used was Adam optimizer, that is a combination of the two famous optimizers RMSprop and AdaGrad and it is a replacement for stochastic gradient descent as it converges faster.

Adam algorithm is an acronym of Adaptive Moment Estimation. It adds a factor to the optimization step, that is the exponential weighted average of the gradients divided by the root of the exponential weighted average of the square of the gradients with an epsilon added to prevent division by 0, and 2 hyperparameters Beta1 multiplied to the numerator and Beta2 to the denominator with default values of 0.9 and 0.999 as shown in figure (6).

$For\ each\ Parameter\ w^j$

$(j\ subscript\ dropped\ for\ clarity)$

$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$\eta : Initial\ Learning\ rate$

$g_t : Gradient\ at\ time\ t\ along\ \omega^j$

$\nu_t : Exponential\ Average\ of\ gradients\ along\ \omega_j$

$s_t : Exponential\ Average\ of\ squares\ of\ gradients\ along\ \omega_j$

$\beta_1, \beta_2 : Hyperparameters$

Figure 6

GRAD-CAM

Gradient Class Activations Mapping (GRAD-CAM):

The GRAD-CAM algorithm is an algorithm that maps class activations that is resulted from the image being fed forward inside the network to the features. As the image gets deeper and deeper inside the convolutional neural network the size shrinks due to convolutions but the number of features increases, and at the last convolution layer it is flattened or average pooled then fed into fully connected layers followed by a classifier layer, so the GRAD-CAM gets the output of the last convolution layer from feeding the network with the image, then it maps it to the prediction of this image, the output features are weighted with the gradients of the output, and finally the resulting output is averaged across the last axis (channels axis) to make it interpretable as an image, so the GRAD-CAM briefly produces a localization heatmap that highlights important regions of the image that affected the decision of the classification.

To show the heatmap, the jet colors were applied to the heatmap, as shown in figure (7) and figure (8), and then the heatmap's size was increased to the size of the original image with the bicubic interpolation algorithm, then the heatmap was overlayed ontop of the original image, and the resulting superimposed image shows the important regions that most affected the decision of the network, and also the heatmap is used for debugging the network if it had made a mistake in the classification to see what regions affected the decision and to take actions upon seeing what made the network predict wrong.

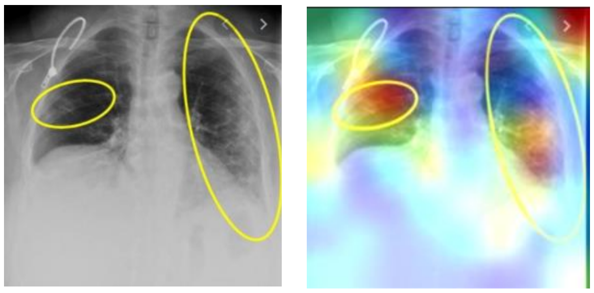


Figure 6

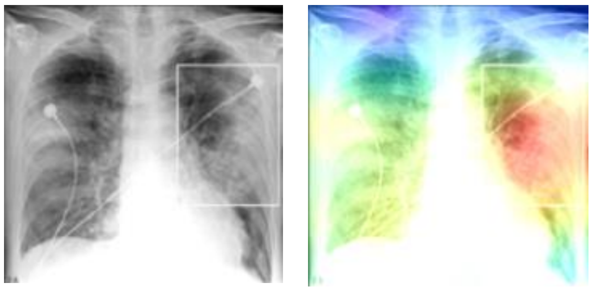
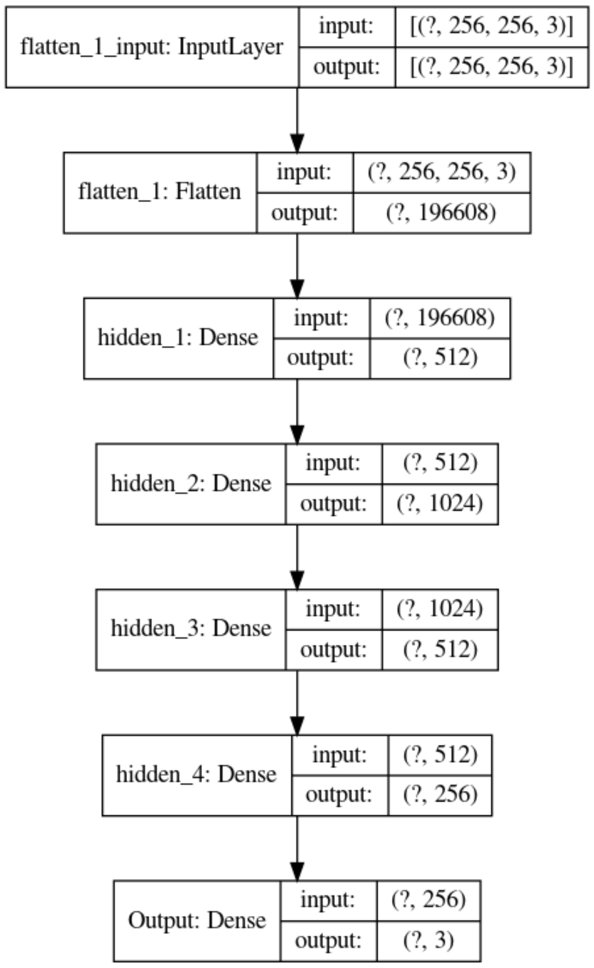


Figure 8

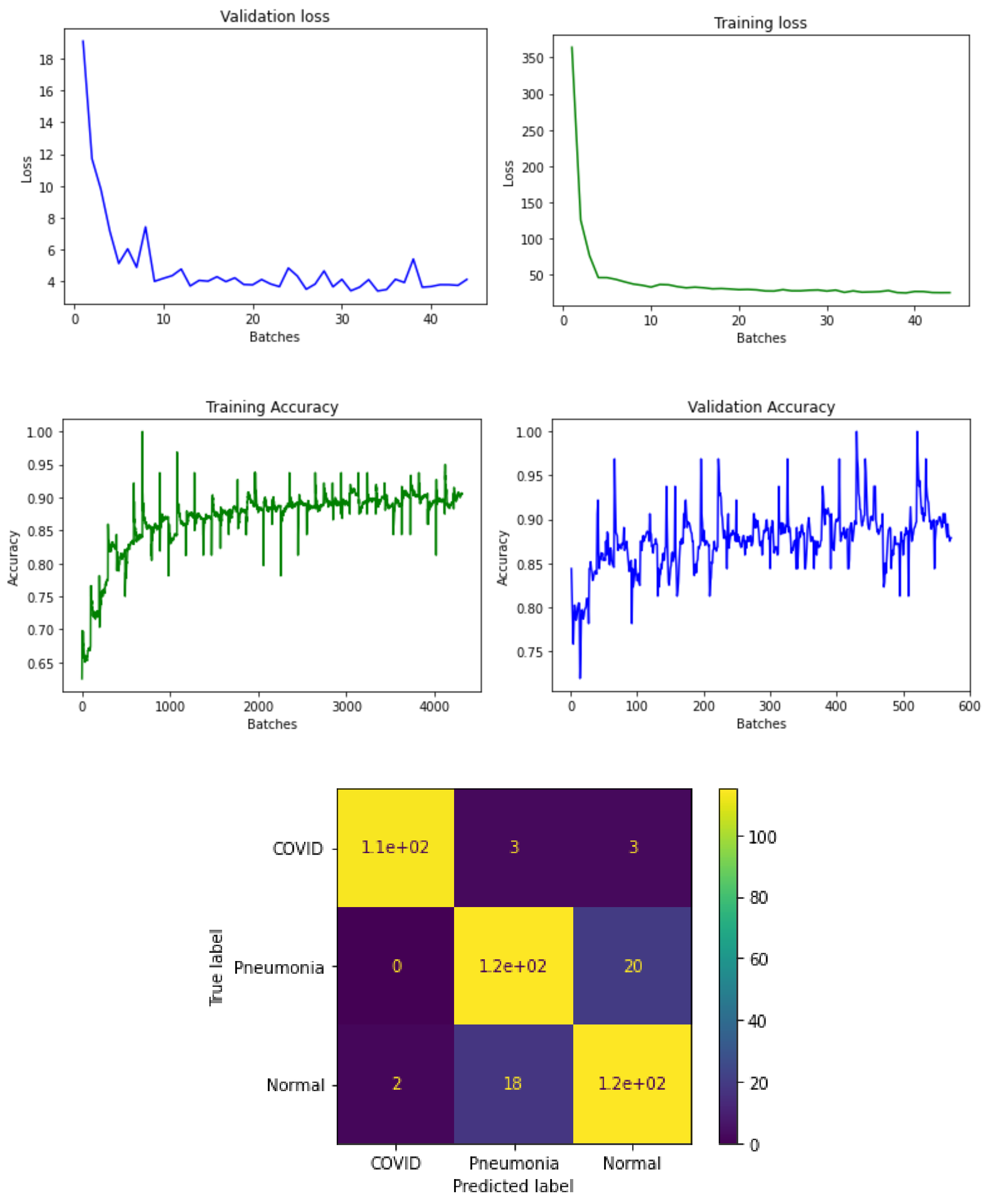
I. Fully-Connected Neural Networks:

a. Shallow Fully Connected Neural Network Model:

A fully connected neural network consists of a series of fully connected layers that connect every neuron in one layer to every neuron in the following layer. The major advantage of fully connected networks is that they are structure agnostic as there are no special assumptions needed to be made about the input. A shallow fully connected neural network is a fully connected network with a minimum number of hidden layers.



Shallow Fully Connected Neural Network Architecture

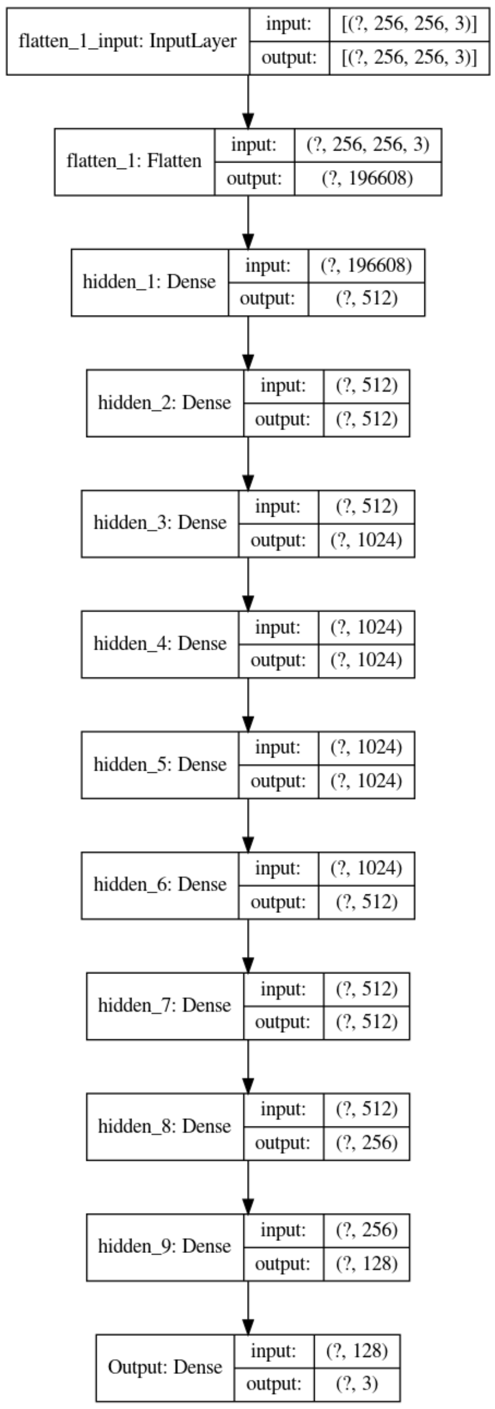


Shallow Fully Connected Neural Network Analysis

b. Deep-Fully Connected Neural Network Model:

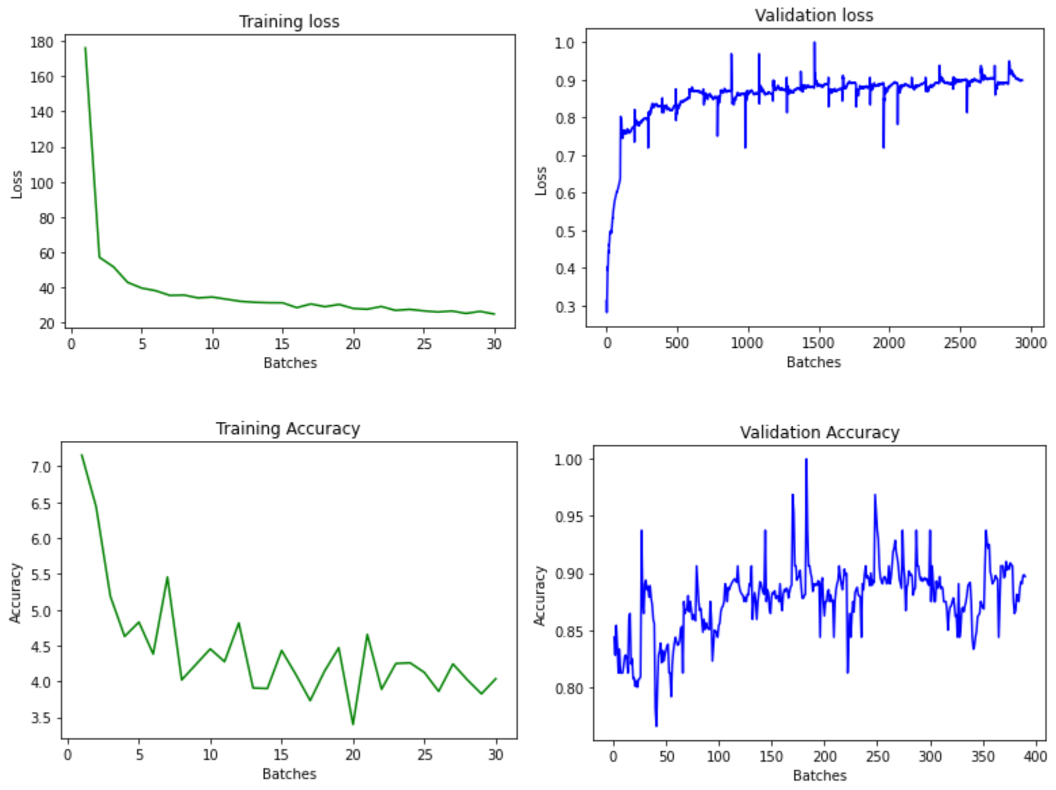
Networks

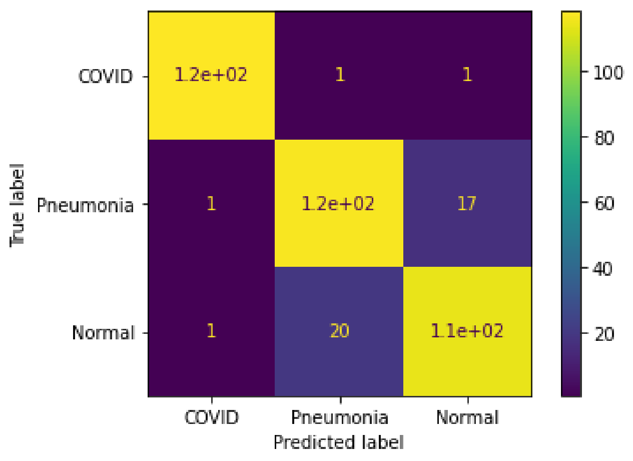
A deep fully connected neural network is a fully connected neural network that contains a relatively large number of hidden layers compared to a shallow model.



flatten_1_input: InputLayer | input: [(?, 256, 256, 3)] | output: [(?, 256, 256, 3)]
flatten_1: Flatten | input: (?, 256, 256, 3) | output: (?, 196608)
hidden_1: Dense | input: (?, 196608) | output: (?, 512)
hidden_2: Dense | input: (?, 512) | output: (?, 512)
hidden_3: Dense | input: (?, 512) | output: (?, 1024)
hidden_4: Dense | input: (?, 1024) | output: (?, 1024)
hidden_5: Dense | input: (?, 1024) | output: (?, 1024)
hidden_6: Dense | input: (?, 1024) | output: (?, 512)
hidden_7: Dense | input: (?, 512) | output: (?, 512)
hidden_8: Dense | input: (?, 512) | output: (?, 256)
hidden_9: Dense | input: (?, 256) | output: (?, 128)
Output: Dense | input: (?, 128) | output: (?, 3)

Deep Fully Connected Neural Network Architecture



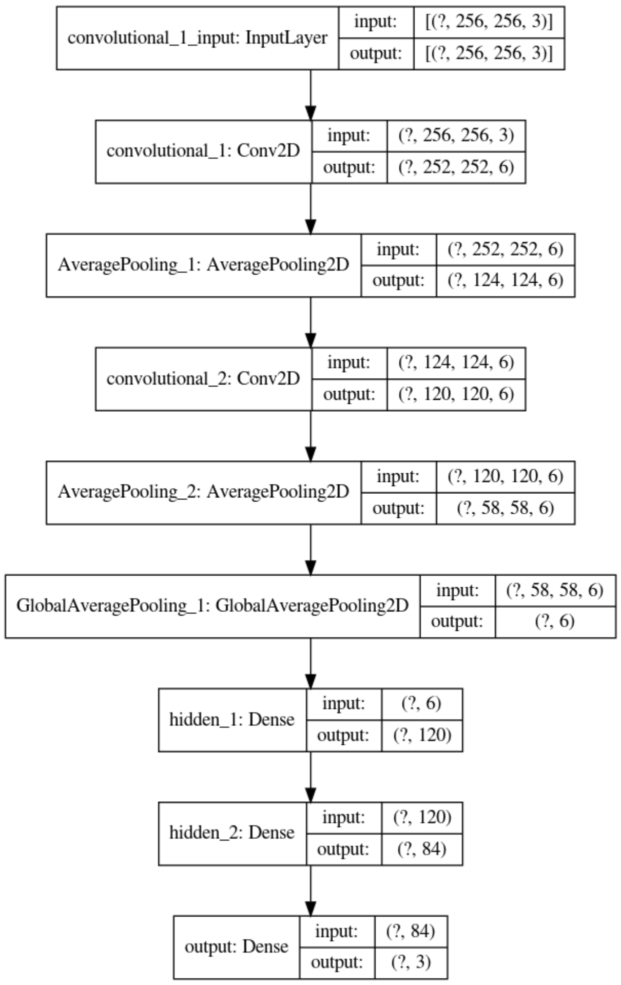


Deep Fully Connected Neural Network Confusion Matrix

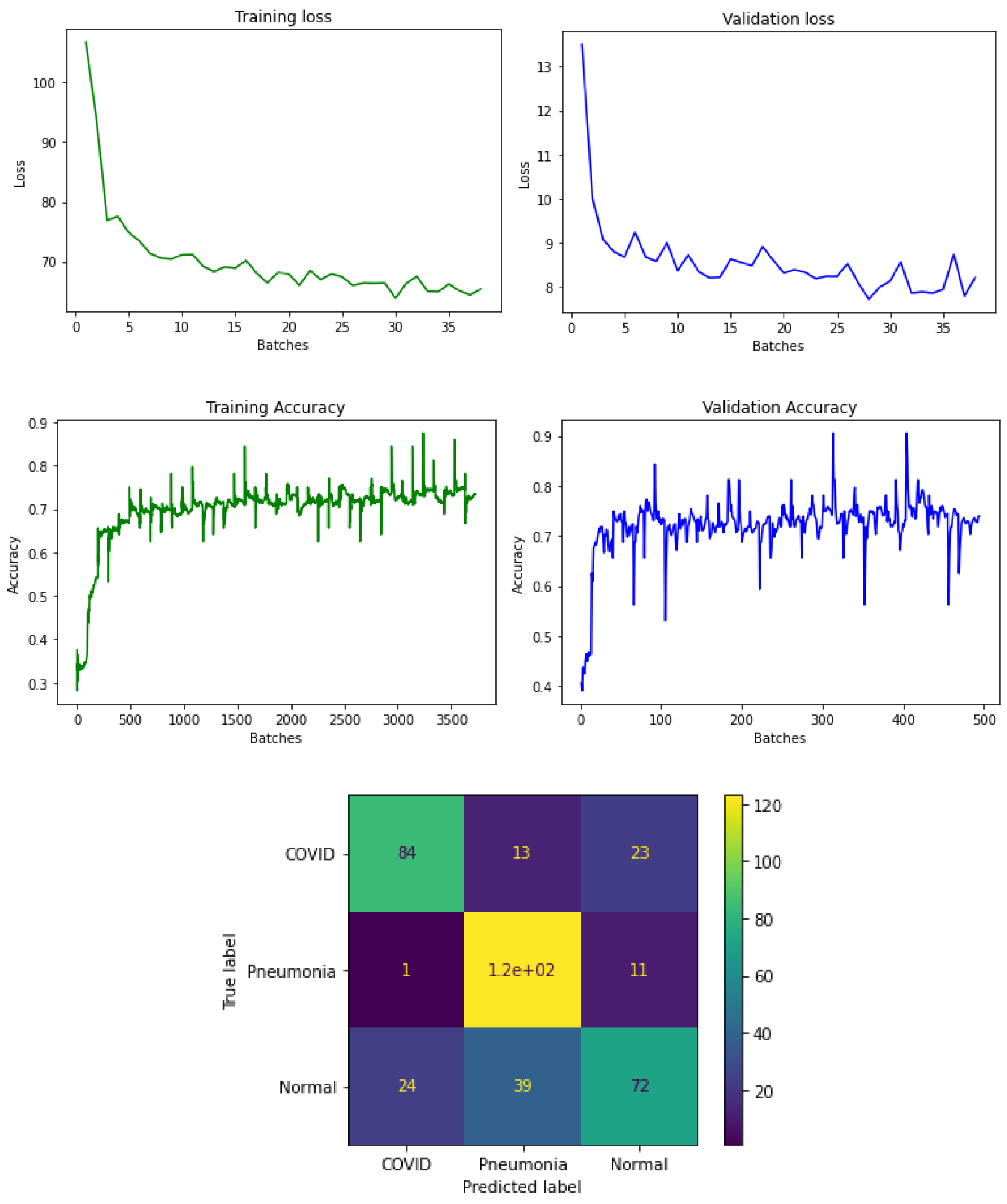
II. Convolutional:

a. Shallow-Convolutional Network Model:

A Convolutional Neural Network (CNN) is a network which can take in an input image, assign importance to various aspects in the image and be able to differentiate one from the other. The pre-processing required in a CNN is much lower as compared to other neural networks. A Shallow CNN consists of a small number of hidden layers.



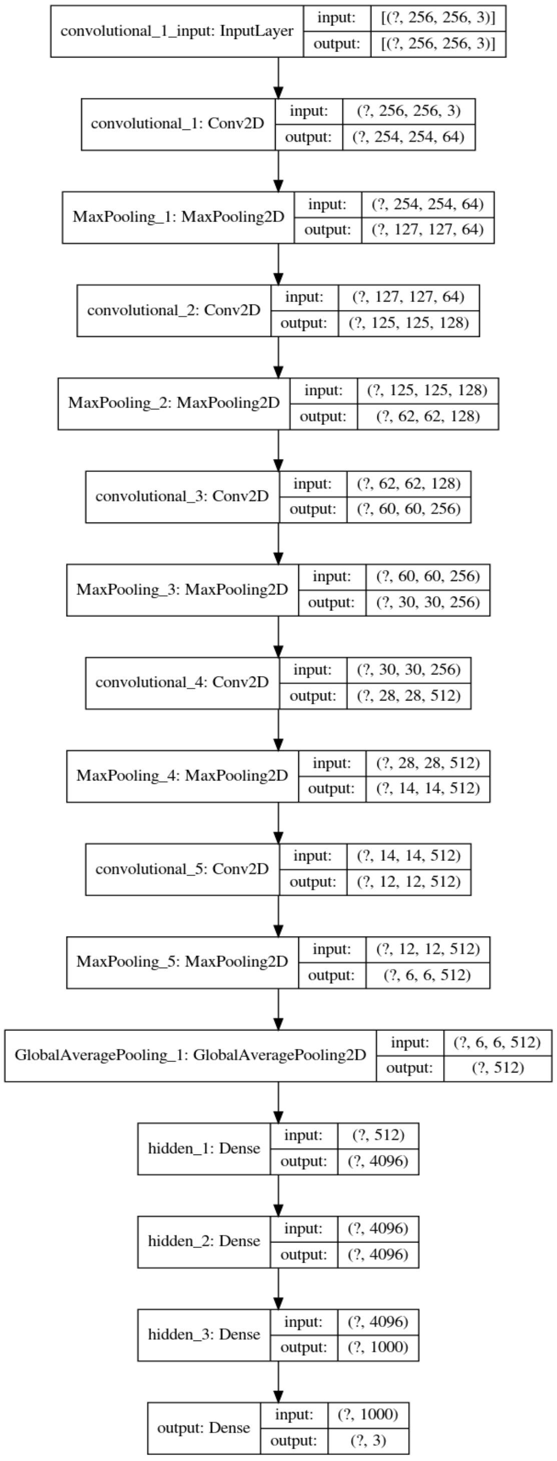
Shallow Convolutional Neural Network Architecture



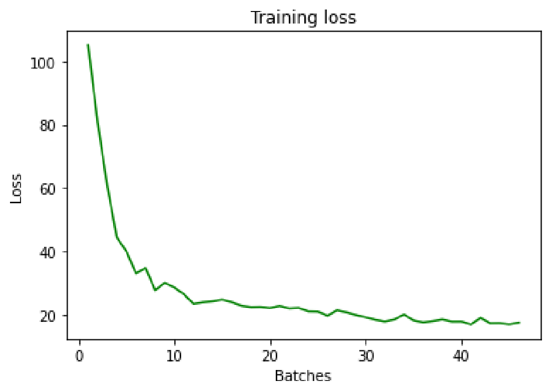
Shallow Convolutional Neural Network Analysis

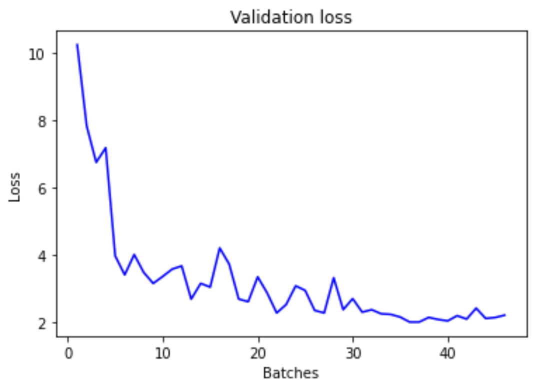
b. Deep-Convolutional Network Model:

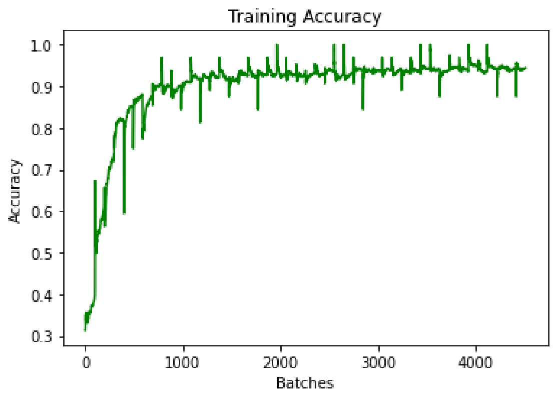
A Deep Convolutional Neural Network is a CNN that has a relatively large number of hidden layers compared to the number of hidden layers in a shallow model.

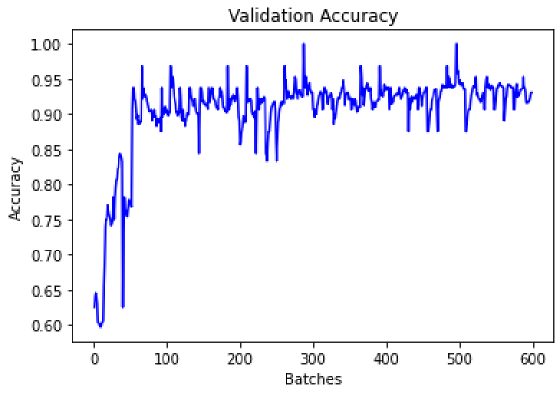

convolutional_1_input: InputLayer | input: [(?, 256, 256, 3)] | output: [(?, 256, 256, 3)]
convolutional_1: Conv2D | input: (?, 256, 256, 3) | output: (?, 254, 254, 64)
MaxPooling_1: MaxPooling2D | input: (?, 254, 254, 64) | output: (?, 127, 127, 64)
convolutional_2: Conv2D | input: (?, 127, 127, 64) | output: (?, 125, 125, 128)
MaxPooling_2: MaxPooling2D | input: (?, 125, 125, 128) | output: (?, 62, 62, 128)
convolutional_3: Conv2D | input: (?, 62, 62, 128) | output: (?, 60, 60, 256)
MaxPooling_3: MaxPooling2D | input: (?, 60, 60, 256) | output: (?, 30, 30, 256)
convolutional_4: Conv2D | input: (?, 30, 30, 256) | output: (?, 28, 28, 512)
MaxPooling_4: MaxPooling2D | input: (?, 28, 28, 512) | output: (?, 14, 14, 512)
convolutional_5: Conv2D | input: (?, 14, 14, 512) | output: (?, 12, 12, 512)
MaxPooling_5: MaxPooling2D | input: (?, 12, 12, 512) | output: (?, 6, 6, 512)
GlobalAveragePooling_1: GlobalAveragePooling2D | input: (?, 6, 6, 512) | output: (?, 512)
hidden_1: Dense | input: (?, 512) | output: (?, 4096)
hidden_2: Dense | input: (?, 4096) | output: (?, 4096)
hidden_3: Dense | input: (?, 4096) | output: (?, 1000)
output: Dense | input: (?, 1000) | output: (?, 3)

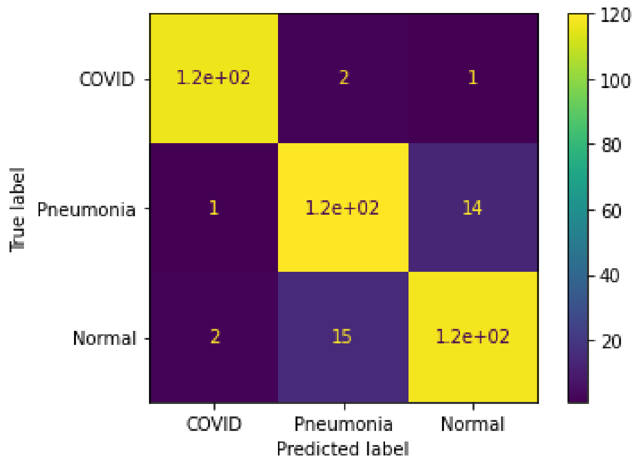
Deep Convolutional Neural Network Architecture









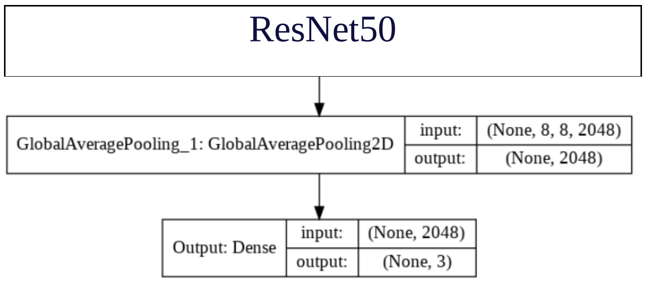


Deep Convolutional Neural Network Analysis

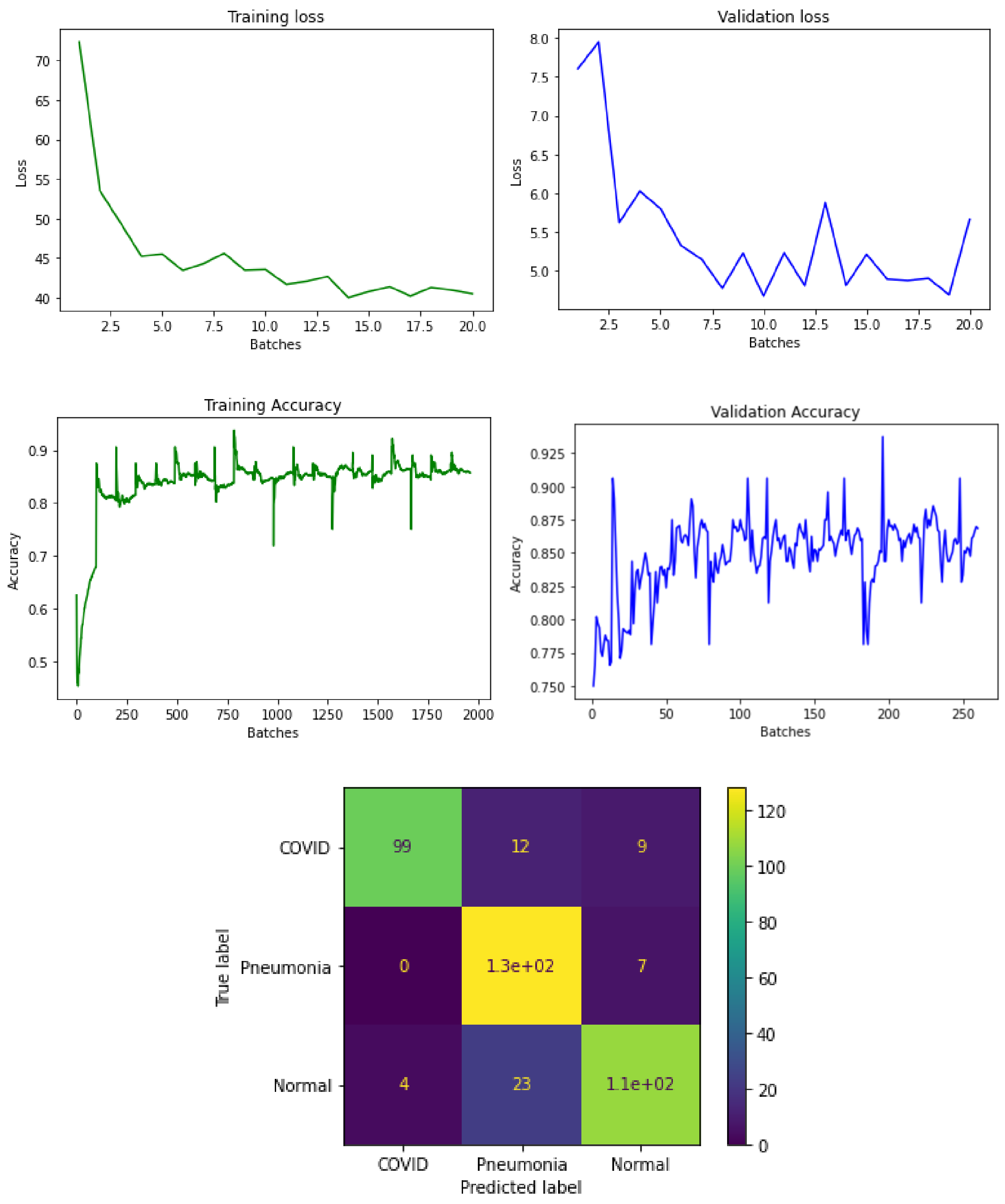
III. Improved Models:

a. ResNet50:

ResNet-50 is a convolutional neural network that is 50 layers deep. The network can be loaded as a pretrained version f itself on more than a million images from the ImageNet database.



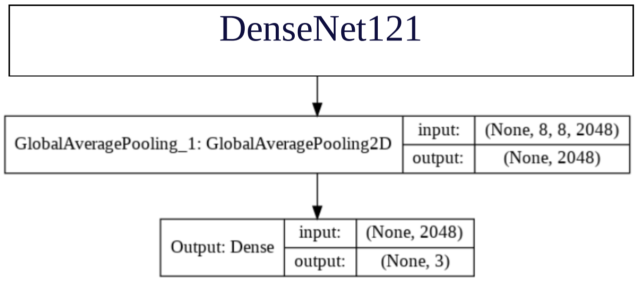
Enhanced ResNet50 Architecture



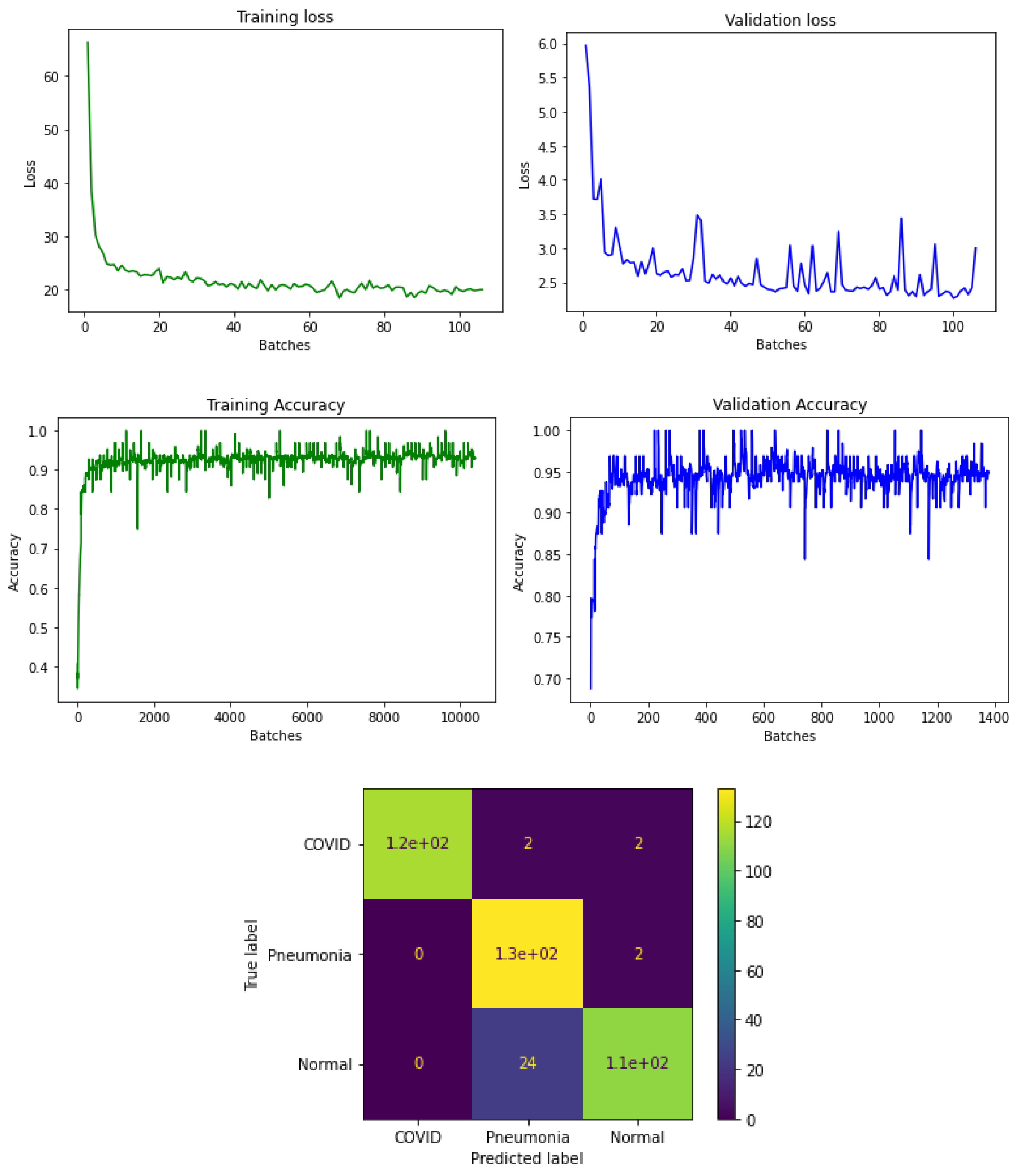
Enhanced ResNet50 Analysis

b. DenseNet121:

DenseNet-121 is a convolutional neural network that is 121 layers deep. The network can be loaded as a pretrained version f itself on more than a million images from the ImageNet database.



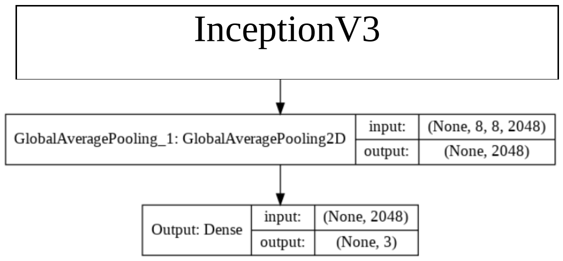
Enhanced DenseNet121 Architecture



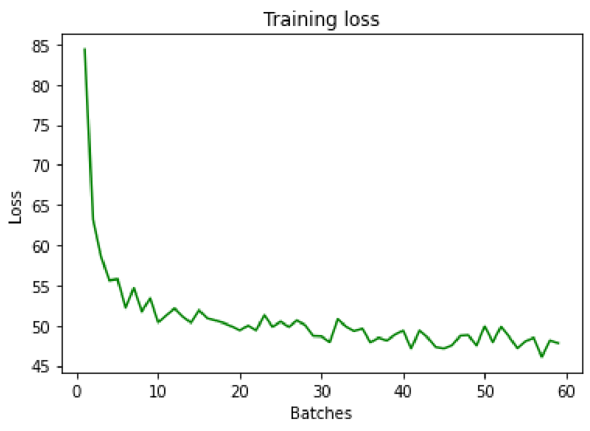
Enhanced DenseNet121 Analysis

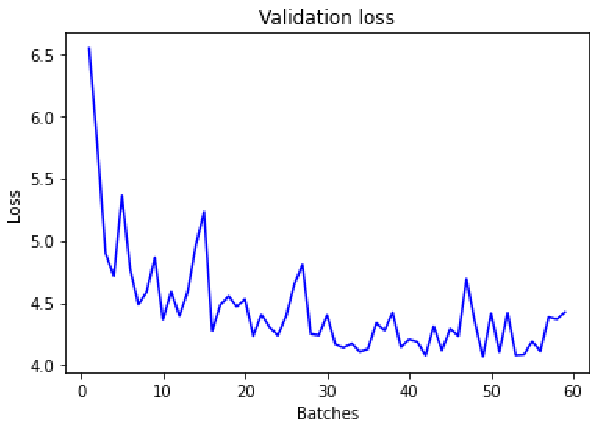
c. InceptionV3:

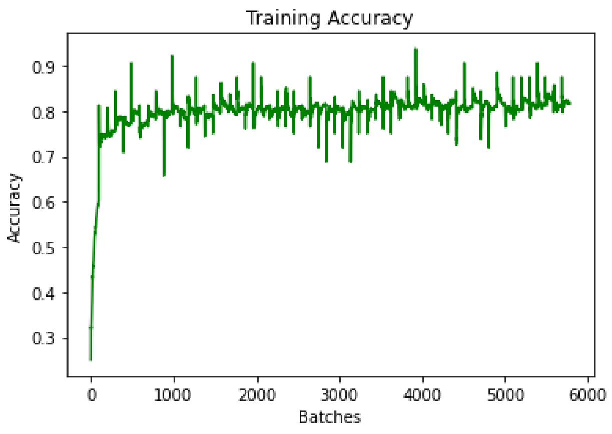
Inception-v3 is a convolutional neural network architecture from the Inception family that makes several improvements including using Label Smoothing, Factorized 7 x 7 convolutions, and the use of an auxiliary classifer to propagate label information lower down the network along with the use of batch normalization for layers in the sidehead. The network can be loaded as a pretrained version f itself on more than a million images from the ImageNet database.

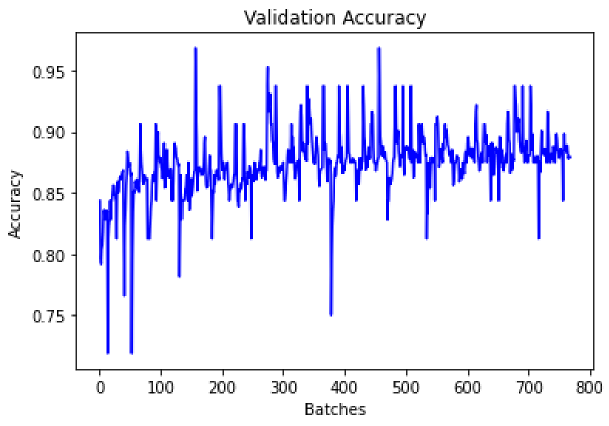


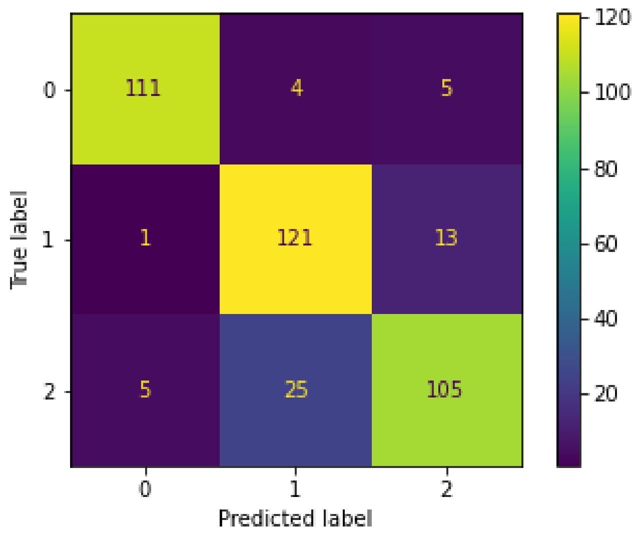
Enhanced InceptionV3 Architecture











Enhanced InceptionV3 Analysis

Analysis

- All models are trained with batch size of 32 for training and 16 for testing over 100 epochs.
- The following tables compares the models after a maximum of 100 epochs:

	Shallow Fully Connected	**Deep Fully Connected**	**Shallow Convolutional**	**Deep Convolutional**	**ResNet50**	**DenseNet 121**	**InceptionV 3**
Training Accuracy	0.9050	0.8986	0.7335	**0.9443**	0.8571	0.9439	0.87709
Training Loss	25.3153	24.9562	65.3884	17.4486	40.5043	**12.9975**	17.8162
Validation Accuracy	0.8788	0.8969	0.73969	0.9304	0.8685	**0.9500**	0.8994
Validation Loss	4.1050	4.0373	8.2188	3.0142	5.6592	**1.6331**	3.5395
Test Accuracy	0.9126	0.9425	0.7468	0.9449	0.8726	**0.9716**	0.8872
Test Loss	7.4688	6.9701	17.4736	5.5667	11.3088	**2.7138**	7.6834
F1 Score (COVID-19)	0.97	0.98	0.73	0.97	0.89	**0.99**	0.92
F1 Score (Pneumonia)	0.85	0.86	0.79	0.88	0.86	**0.94**	0.83
F1 Score (Normal)	0.84	0.85	0.60	0.88	0.83	**0.93**	0.82