

## Project Report

on

# **COVID-19 Pneumonia Detection using CNN**

*by*

*Kasturi Joshi*

*UIN 662904838*

*kjoshi27@uic.edu*

*Utsav Bhadresh Shah*

*UIN 663598526*

*ushah21@uic.edu*

Computer Science Department

at

University of Illinois at Chicago

Date : May 8, 2020

## INDEX

Sr No	Topic	Page No
1	Introduction	3
2	Problem Definition	3
3	Dataset	3
4	Pre-processing	4
5	Related Work	6
6	Methodology	7
7	Results	17
8	Conclusion	17
9	Future Work	18
10	Bibliography	19

## 1. Introduction

The novel CoronaVirus (COVID-19) has infected millions of people around the world. This pandemic has not only infected people and taken the lives of lakhs, but also affected people's life in a really harsh manner where-in all trivial daily activities have become a luxury. It has infected over a million people in the US and the death toll is very high. This pandemic has taken a big toll on the medical community. The nurses and doctors are the ones whose life has been affected the most. The purpose of this project is to help the global medical community in easing their work through Machine Learning and make the process of COVID-19 Pneumonia diagnosis faster and error free, in order to avoid false negatives and save the lives of many by providing quick treatment.

## 2. Problem Definition

COVID-19 affects the respiratory system of the human body, and one of its complications is the pneumonia that is caused due to it, which can be life threatening. This pneumonia is detected using the X-rays of the patients. However, the difference between COVID-19 pneumonia and viral pneumonia is very difficult, even for medical experts to scan visually.

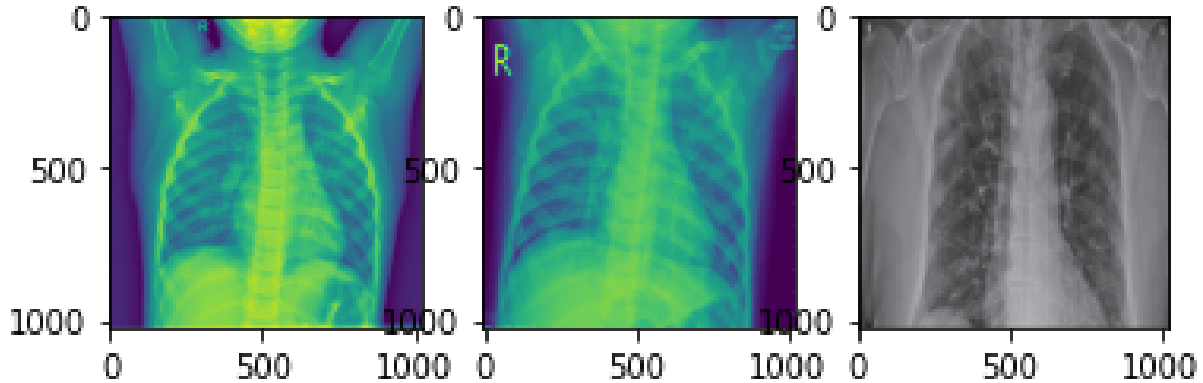
To help eliminate this problem, we aim to build a Deep Learning model that distinguishes the x-ray images as COVID-19 pneumonia, viral pneumonia and a healthy patient. To implement this, we have used **Convolution Neural Networks (CNN)** on the patient's chest X-Ray images. We have also implemented **Support Vector Machine** classifier and **Multi-Layer Perceptron** as baseline classifiers to our CNN model. We aim to maximize our accuracy using the different methods mentioned above, because wrong diagnosis in the medical community could cause a huge impact and could even be fatal.

## 3. Dataset

We used the COVID-19 radiography dataset from kaggle[1]. The dataset is open source and can be used by anyone. It contains chest x-ray images of patients. The data is formatted such that it contains folders for each of the three classes and metadata files giving attributes of the data such as name, file type(PNG), file path and image size in pixels. The dataset does not have any CSV file with image data along with their labels.

The dataset contains 219 images of COVID-19 patient's lungs, 1341 images of healthy patient's images and 1345 images of viral pneumonia patients. These images are 1024 X 1024 pixels with all three channels that are RGB.

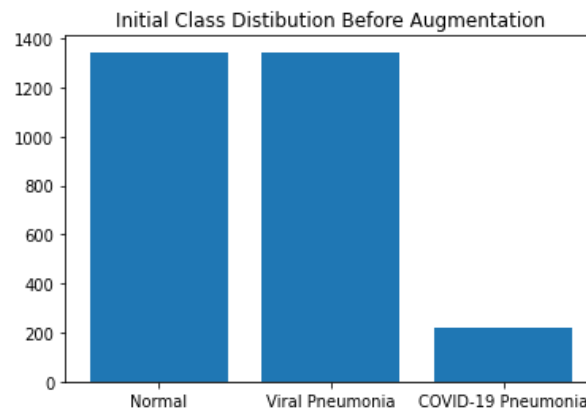
The images look like this:



*Fig 3.1: Figure showing X-ray images for Normal, Viral and COVID-19 Pneumonia*

#### 4. Pre-processing

The dataset has **class imbalance**; there are more images for normal and viral pneumonia patients than COVID-19 patients, as can be seen from the figure below.

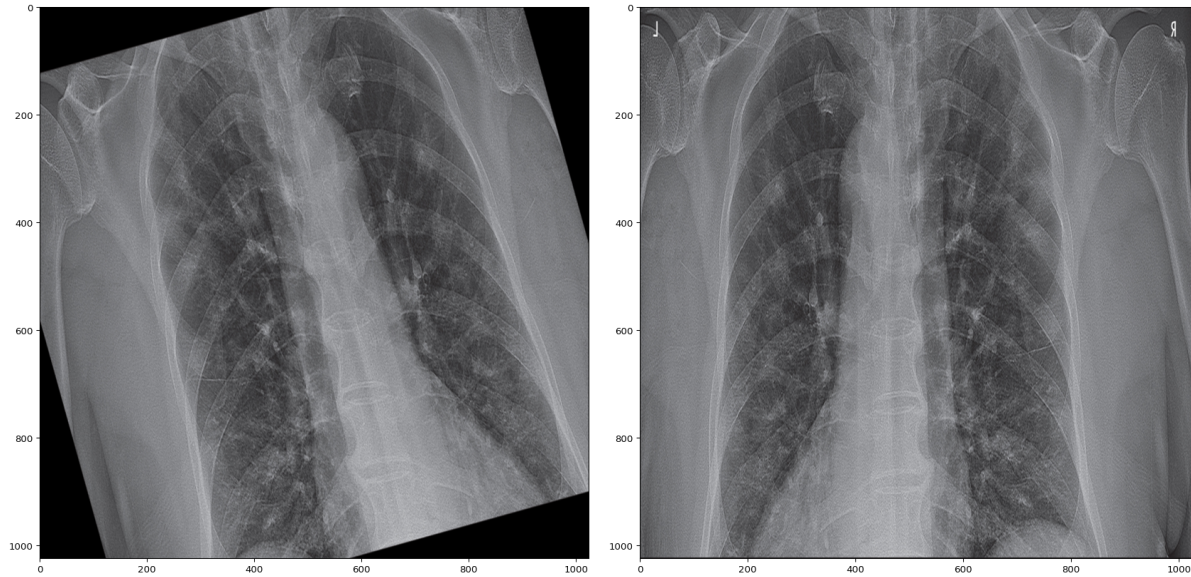


*Fig 4.1: Figure showing initial class distribution*

Data imbalance is a major problem in Machine Learning and in order to counter this problem, **data augmentation** techniques were used to increase the number of COVID-19 images in the dataset. There are multiple data augmentation methods that can be applied to increase the dataset like rotation, contrast change, shearing, zooming, cropping, mirroring, adding noise, etc.

We chose to take majorly **rotation, mirroring and contrast** change. The reason for choosing only these techniques is that other techniques change the input image drastically which basically makes the image not look like lung X- Rays.

The augmented images were saved back and the final dataset had COVID-19 images increased from 219 to 1293 which is almost the same size as other classes. This solved the data imbalance problem. The image augmentation example is as follows:



*Fig 4.2: Figure showing COVID-19 patient's X-ray images rotated and mirrored*

The data was split into **3 subset** viz. **training, validation and testing**. The testing data was set aside before augmenting the data so that only original data is used to test the models. The **testing** data includes 40 images from each of the classes making it a total of **120** images. The **validation** set contains 60 images from each of the classes making it a total of **160** images. The rest are **training images** which total to **3679**.

After the data augmentation and train, validation and test split, all the data was saved in separate CSV files each containing those individual splits. The CSV file had the file path and a label for that particular image. The classes were **encoded** making labeling COVID-19 as 0, Normal as 1 and Viral pneumonia as 2.

	filepath	label
0	/content/COVID-19 Radiography Database/Viral P...	2
1	/content/COVID-19 Radiography Database/NORMAL/...	1
2	/content/COVID-19 Radiography Database/Viral P...	2
3	/content/COVID-19 Radiography Database/NORMAL/...	1
4	/content/COVID-19 Radiography Database/Viral P...	2

*Fig 4.3: Figure showing data csv*

Data Preprocessing is required before passing the data into the SVM, MLP and CNN models. The original image is a 1024 X 1024 pixel image. SVM and MLP expect a numpy array containing a single row of values[2][3]. The pre-trained CNN models expect an image input of 224X 224 X 3 indicating that the input image should be **rescaled** to 224 pixels and should have 3 channels[4]. The input data was shuffled, rescaled and preprocessed using inbuilt keras libraries. The preprocessing includes normalizing the images before feeding them into the network.

## 5. Related Work

Muhammad et al[8] used the COVID-19 kaggle dataset to detect the presence of COVID-19 pneumonia. They performed data augmentation to generate 2600 images per class and used SqueezeNet, ResNet-18, DenseNet201 and AlexNet models. They achieved an accuracy of 98.3% and sensitivity, specificity and precision of 96.7%, 100%, 100% and 98.3%, 96.7%, 99% and 100% respectively.

Minaee et al[9] also implemented the same four CNN architectures as [8] for this problem. They used Pytorch for their implementation. They achieved a sensitivity rate of 97%(± 5%), while a specificity rate of around 90%.

Asmaa et al[3] implemented their previously developed DeTraC CNN model to classify COVID-19 x-ray images. they achieved an accuracy of 95.12%.

Sethy et al[2] implemented a model which consisted of resnet50 plus SVM to detect the presence of COVID-19 pneumonia. They achieved an accuracy of 95.38% and FPR, F1 score, MCC and Kappa score of 95.52%, 91.41% and 90.76%.

Xianghong et al. [15] built a customized VGG16 model for lung regions identification and different types of pneumonia classification.

Shuai et al. [16] used deep learning techniques on CT images to detect COVID-19 patients with an accuracy, specificity and sensitivity of 89.5%, 88% and 87% respectively.

Linda et al. [17] introduced a deep convolutional neural network, called COVID-Net for the detection of COVID-19 cases from the chest X-ray images with an accuracy of 83.5%.

Novelty - Our approach is different from all the approaches mentioned above. We have used both deep learning and traditional machine learning algorithms. None of these papers have used the VGG-19 model to detect COVID-19 pneumonia. Our accuracy, 97.5% on the VGG-19 model, for COVID-19 screening is higher than most of the papers we have surveyed above, except [8]. We have tuned the parameters with our own knowledge and understanding of convolutional neural networks. We tried implementing multiple models and learning from each trail we improved the next model to make it fit better with our data.

## 6. Methodology

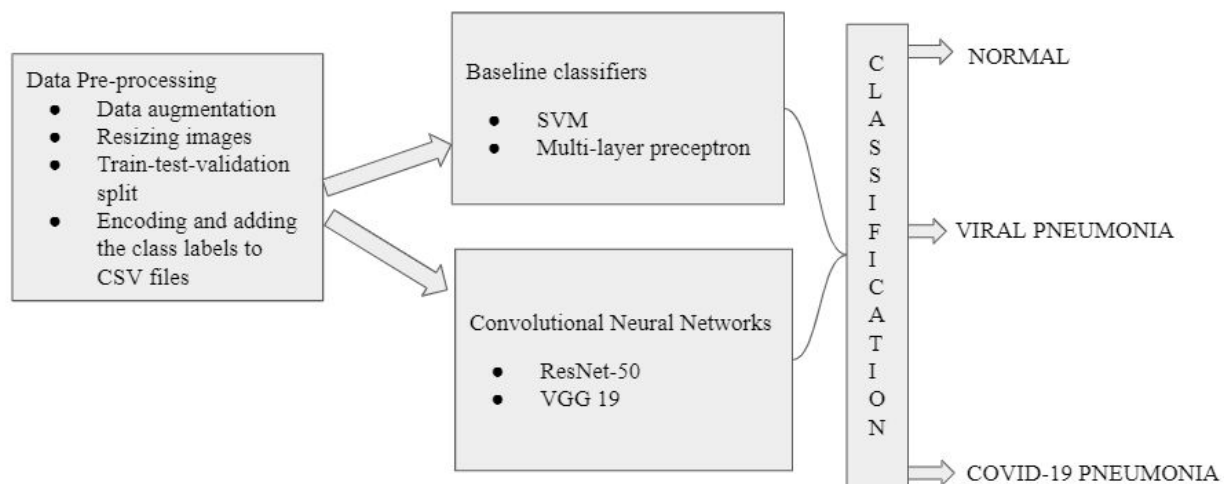


Figure 6.1 Figure showing the COVID-19 detection model pipeline

We implemented four different machine learning and deep learning algorithms to classify our x-ray images into the three categories mentioned [Figure 6.1]. The dataset was split into training and testing data. We implemented a traditional machine learning algorithm, Support Vector Machine (SVM), as our baseline classifier. To further improve the accuracy, we implemented Multi-Layer Perceptron, which is a neural network. The two other algorithms we implemented were the ConvNet algorithms namely, **RESNET-50** and **VGG-19**. The **transfer learning** models, the pre-trained models from the ImageNet dataset, for these two algorithms were used in order to build more accurate models and save time. All the algorithms and their results will be discussed in the further paragraphs.

## 6.1 Pre-processing for transfer learning

The image size required for the pre-trained RESNET-50 and VGG-19 model is 224,224,3. So, this size was kept as the standard input size for all the algorithms. Keras image preprocessing library was used to load the images and resize them to 224,224.

## 6.2 Support Vector Machine (SVM) Classifier

The SVM classifier was implemented to ensure that the data is linearly separable, as precisely as possible, as it is necessary while working with a medical dataset. Also, SVM allows us to try multiple kernels and evaluate which one gives a good accuracy. The classifier does a good job to classify the data into the 3 classes.

The SVM model requires an input of **n\_samples, n\_features**. The input images were preprocessed to convert them to the required format. The image loaded as 224,224,3 was transposed and reshaped to be 1,150528. Here, 1 is the number of samples and 150528 are the total number of pixels in a 224,224 image. All the 3679 images were converted to this dimension and concatenated to give a 3679,150528 input array. Similarly, the testing images were converted to 120,150528 shaped vector. The labels are converted to numpy arrays and passed in for training. The array looks like this, [0,1,1,2,0,2,...,0,1]. For this particular dataset we used SVM with an rbf kernel and tuned to have a C value of 100. The SVM implementation was done using sklearn library[2]. The type of kernel and the value of C was tuned by doing multiple iterations with different values for the same. The final model looked as follows[Figure 6.2.1]:

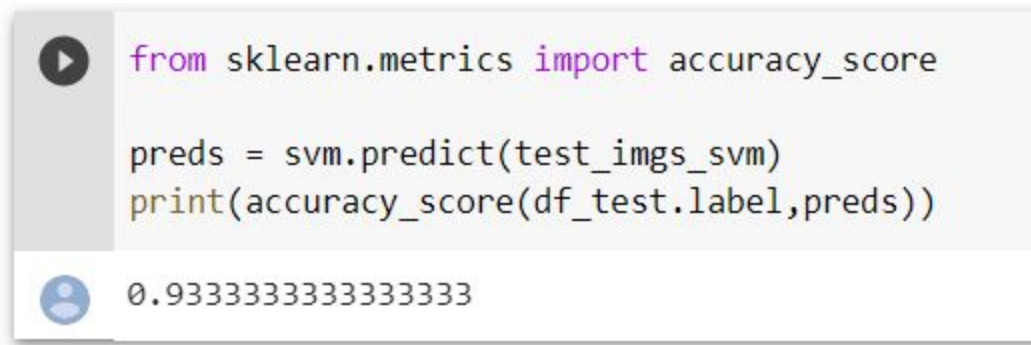
```
svm.fit(train_imgs_svm, df_train.label)

SVC(C=100, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

*Fig 6.2.1: Figure showing SVM model parameters*

The model was trained and tested against the test set to give an accuracy of 93.33% which is a pretty good model given the complexity of the data and the high dimensionality[Figure 6.2.3].





```
from sklearn.metrics import accuracy_score

preds = svm.predict(test_imgs_svm)
print(accuracy_score(df_test.label,preds))
```

0.9333333333333333

*Fig 6.2.3: Figure showing SVM model's test accuracy*

The reason for not doing a K-Fold cross validation is that each model training took 20-30 minutes. So, a 10 fold cross validation would go on for hours. Also, we had separated the test dataset from the original data before augmentation which is what the real world data would look like. A K-Fold cross validation would contain rotated and other augmented images in it's test set which is not what real world data would look like.

### 6.3 Multi-layer Perceptron

The Multi-Layer perceptron is one step ahead in terms of learning complex data than SVM. That made it our second choice. The MLP algorithm also expects an input of `n_samples, n_features`. So, the same input vector and test vector used for SVM was used for training and testing resp. in the MLP algorithm. The MLP algorithm was tuned by using different number of layers, different number of nodes, different solvers, different learning rate and other hyperparameters. The momentum method was implemented to speed up the learning of the MLP. The inclusion of a momentum term has been found to increase the rate of convergence dramatically[5][7]. Early stopping is widely used to avoid overfitting the data and stop the training with the minimum loss value for the network[6]. We implemented **early-stopping** to stop the training of the algorithm if there is no decrease in the loss function in 10 consecutive epochs. This makes sure that the most optimum model is achieved. The learning rate parameter's default value is being constant throughout the network and for all the epochs of the training. For our case, we used adaptive learning rate. If the loss function's value does not improve, the learning rate's value is reduced by some amount. This helps the function to minimize quickly when the loss function is continuously decreasing but also does a finer tune to not skip the minima[7]. This is used when the solver is Stochastic Gradient Descent. This solver was also a parameter which was experimented upon.

We tried training the MLP network started with **4 layers** with each having a **100 nodes**. This network achieved an accuracy of **89.33**. We then tried a more complex network with **7 layers**

**and 100 nodes** each in all the layers. This network was overfitting the data, as the testing accuracy was 82.5%. The final configuration that gave the best accuracy is with **6 layers, 100 nodes** in each layer, with **relu** activation function. We got the best accuracy for the model with a learning **rate** of **0.0001**, **adam optimizer**[Figure 6.3.1].

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=True, epsilon=1e-08,
              hidden_layer_sizes=(100, 100, 100, 100, 100, 100),
              learning_rate='adaptive', learning_rate_init=0.001, max_fun=15000,
              max_iter=10000, momentum=0.9, n_iter_no_change=10,
              nesterovs_momentum=True, power_t=0.5, random_state=None,
              shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
              verbose=False, warm_start=False)
```

*Fig 6.3.1: Figure showing MLP model parameters*

This model was trained and tested and gave maximum testing accuracy of **92.5%** which is less compared to a neural network[Figure 6.3.2]. We believe a further tuning of the MLP would help us achieve better accuracy but it serves as a good baseline estimator. Our aim was to achieve accuracy greater than 97% which could not be achieved without using CNN. So, we shifted to tuning the CNNs rather than investing more time in tuning the MLP to achieve greater accuracy.

```
from sklearn.metrics import accuracy_score
preds = model_nn.predict(test_imgs_svm)
print(accuracy_score(df_test.label,preds))

0.925
```

*Fig 6.3.2: Figure showing MLP model's testing accuracy*

## 6.4 Transfer Learning

The deep learning models take days and weeks to train and optimize weights. Researchers have invested time and resources to train their models and build state-of-the-art systems. Transfer learning is when a model that has been trained on a certain dataset and it gives really good performance, it's weights are frozen and these trained weights are used to solve problems of other datasets. The deep learning models are generally trained on the ImageNet dataset to classify the dataset into 1000 different classes. These models have parameters that are trained

based on the dataset and the final weights are the one that recognize the best features required to perform classification. When a new problem is given, we can initialize with the imagenet weights and edit the final softmax function based on our number of classes or edit the fully connected layers at the back to learn to classify on our custom dataset. The majority chunk of the weights are adopted from the imagenet dataset and kept frozen, as the number of parameters to train now are very less, the classification problem can be solved more quickly.

Transfer learning is used with CNNs and typically other deep neural networks. The reason is too many trainable parameters for the original network. For example, a VGG-19 model has a total of over 26 million trainable weights and in order to get the optimum weights by learning from scratch would take a lot of resources, a lot of time and a lot of data which generally very few people have. So, by using transfer learning and initializing with imagenet weights, just by editing the final fully connected layer and keeping those weights as trainable, the weights to be trained reduce from 26 million to around 6 million. This drastically reduces the training time.

One can imagine using an image classification model trained on ImageNet (which contains millions of labeled images) to initiate task-specific learning for COVID-19 detection on a smaller dataset. Transfer learning is mainly useful for tasks where enough training samples are not available to train a model from scratch, such as medical image classification for rare or emerging diseases, in which sufficiently large numbers of labeled samples may not be available. This is especially the case for models based on deep neural networks, which have a large number of parameters to train. By using transfer learning, the model parameters start with a ready-good initial value that only needs some small modifications to be better curated toward the new task [9].

#### **6.4.1 ResNet-50**

ResNet-50 is a 50 layer deep residual convolutional neural network. It is widely used in solving classification problems using image data. The ResNet-50 model gave an accuracy of 98% for the classification task of COVID-19 pneumonia, viral pneumonia and normal patients in paper [10] and we tried to replicate those results. However, the parameters were not mentioned in the paper and we tried tuning our hyperparameter to achieve different results. The ResNet has special characteristics which makes it better than the other types of convolutional neural networks. It basically solves the problem of vanishing and exploding gradients. With deep neural networks, as we increase the number of layers, the gradients tend to vanish or explode, causing unexpectedly small weight updates. In case of vanishing gradient, the weights get updated with extremely negligible values and in case of exploding gradient, it gets updated by very large values, both of them hinders the appropriate training of the model.

This problem can be overcome using Residual Networks like ResNet. The residual networks have skip connections that basically transfer the layer output not to the next layer but the layer after it. The skipped layers help to reduce the number of trainable parameters significantly,

propagate features to further layers and solve the vanishing gradient problem. All this makes the network more robust and better suited for training.

The ResNet-50 was initialized and implemented from the keras implementation[4]. We initialized the imagenet pretrained weights for the ResNet-50 architecture that we implemented. The initial layers were frozen, that is their weights will not be updated. We removed the last fully connected layer and updated the network with our custom fully connected layers. In order to further make sure that the network does not overfit, we added dropout and regularization after each fully connected layer to make the network more robust[Figure 6.4.1.1].

```

model_resnet = Sequential()
model_resnet.add(resnet)
model_resnet.add(Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model_resnet.add(Dropout(0.3))
model_resnet.add(Dense(3, activation='softmax', kernel_regularizer=regularizers.l2(0.01)))
model_resnet.compile(loss='categorical_crossentropy',
                    optimizer=optimizers.RMSprop(learning_rate=0.0001),
                    metrics=['accuracy'])
# model_resnet.summary()

```

*Fig 6.4.1.1: Figure showing ResNet model's Fully Connected Layers*

We added a dense layer with **512** nodes and the final layer with **3 neurons** and a **softmax** activation function in the output layer to give probabilities for each of the 3 classes. We chose this final model after a lot of parameter tuning, which is mentioned in the next paragraph. However, the output layer was common in all the models we tried. Our network differed in the number of neurons in the fully connected layer and the number of such layers. The **dropout** was kept to **0.3** that is out of all the neurons in that layer, 30% will be dropped in each epoch to make the network robust. Each training instance was run for **100 epochs** and we used **early stopping** to stop and **save the best model** out of all the epochs[Figure 6.4.1.2].

We started off by using **2 layers with 512 nodes** each followed by the softmax layer, this architecture was giving a **99.9** percent accuracy for the **training** dataset but a **validation** accuracy of around **80%**. The **testing** accuracy came out to be **75%** which implied that the network was overfitting on the data. So, we added **l2 regularization** to avoid this problem. This time the accuracy increased but not significantly. The testing accuracy went up to **80%**. We experimented with different **optimizers** like **RMSprop**, **adam**, **Stochastic Gradient Descent** (SGD) and their different learning parameters ranging from 0.001 to  $1 \times 10^{-7}$ .

```

▶ from datetime import datetime
  from keras.callbacks import ModelCheckpoint

  checkpointer = ModelCheckpoint(filepath='/model/weights.h5', verbose=1, save_best_only=True)
  |
  # datetime object containing current date and time
  now = datetime.now()

  dt_string = now.strftime("%d-%m-%Y-%H-%M-%S")
  print("now =", dt_string)

  history = model_resnet.fit(x=train_imgs[0],
                            y=y_train,
                            epochs=100,
                            validation_data=(val_imgs[0],y_val),
                            shuffle=True,
                            callbacks=[checker],
                            verbose=1)

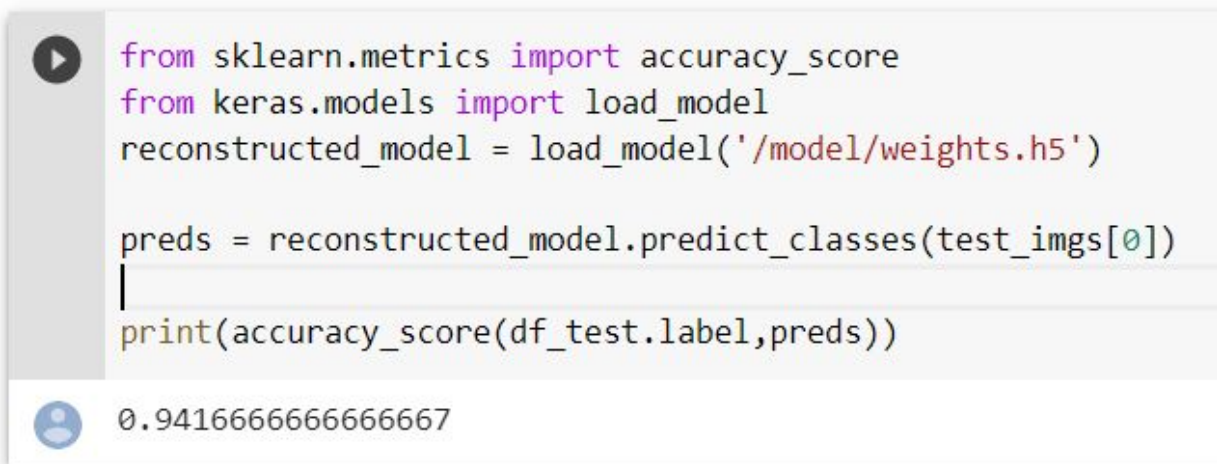
  # model_resnet.save('/model/resnet50_model_'+dt_string+'.h5')
  print("done")

```

*Fig 6.4.1.2: Figure showing the training attributes of the ResNet model*

The other networks that we tried did not achieve sufficient accuracy with only one fully connected layer of 1024 nodes or one layer of 256 nodes followed by the softmax function. All these architectures did not give an accuracy above 85%.

The best architecture for ResNet-50 was with weights frozen to imagenet, the final fully connected layer having one layer of **512** nodes and a final layer consisting of **3 nodes** with a **softmax** function. The optimizer used was **RMSprop** with a **learning rate of 0.0001** with **l2 regularizer** parameter set to **0.01**. The fully connected layer had a **relu** activation function and **loss** set to **categorical cross entropy**[Figure 6.4.1.1]. This network was trained for 100 epochs and the model with the least loss function value was selected as the best model. This best model showed an accuracy of **94.166%**[Figure 6.4.1.3]. The **training time** of our ResNet-50 model was **1 hour and 7 minutes**.



```
from sklearn.metrics import accuracy_score
from keras.models import load_model
reconstructed_model = load_model('/model/weights.h5')

preds = reconstructed_model.predict_classes(test_imgs[0])
|
print(accuracy_score(df_test.label, preds))
```

0.9416666666666667

*Fig 6.4.1.3: Figure showing testing accuracy of the ResNet model*

The ResNet-50 model performed decently well on the dataset but it was overfitting the data as we had a small dataset. Thus, we thought of trying a less deeper CNN architecture, in order to fit our data well and give more accurate predictions.

### 6.4.2 VGG-19

VGG-19 is a 19 layer deep neural network whose name comes from the team who developed and published this model to be ranked on the ILSVRC classification task. The VGG-19 model was ranked second in the competition which made this as one of the best models around 2015[12].

The VGG-19 has 16 layers of convolution and pooling and 2 fully connected layers followed by a softmax layer. For our implementation, the model was initialized and implemented from the keras implementation[4]. We initialized the imagenet pretrained weights for VGG-19 architecture that we implemented. The initial layers were frozen. We removed the last fully connected layer and updated the network with our custom fully connected layers. In order to further make sure that the network does not overfit, we added dropout and regularization after each fully connected layer to make the network more robust and to make sure that it does not overfit[Figure 6.4.2.1].



```

model_vgg19 = Sequential()
model_vgg19.add(vgg19)
model_vgg19.add(Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model_vgg19.add(Dropout(0.3))
model_vgg19.add(Dense(3, activation='softmax', kernel_regularizer=regularizers.l2(0.01)))
model_vgg19.compile(loss='categorical_crossentropy',
                    optimizer=optimizers.RMSprop(learning_rate=0.0001),
                    metrics=['accuracy'])
model_vgg19.summary()

```

*Fig 6.4.2.1: Figure showing the VGG-19 model's Fully Connected Layers*

We added a dense layer with **tunable** nodes and the final layer with **3 neurons** and a **softmax** activation function in the output layer to give probabilities for each of the 3 classes. We chose this final model after a lot of parameter tuning, which is mentioned in the next paragraph.

However, the output layer was common in all the models we tried. Our network differed in the number of neurons in the fully connected layer and the number of such layers. The **dropout** was kept to **0.3**. Each training instance was run for **50 epochs** and we used **early stopping** to stop and **save the best model** out of all the epochs [Figure 6.4.2.2]. L2 Regularization was also implemented as a learning from our ResNet implementation. Each of the models took about 25-30 minutes to train for 50 epochs even with a GPU instance.

We started off by using **1 layer with 512 nodes** followed by the softmax layer. This architecture was giving a **98%** percent accuracy for the **training** dataset and a **validation** accuracy of around **95% for the best model**. The **testing** accuracy came out to be **94.6%** which implied that the network was not a perfect fit on the data. We experimented with different **optimizers** like **RMSprop** and **adam** and their different learning parameters ranging from 0.001 to  $1 \times 10^{-7}$ . We then tried to increase the complexity of the model to make sure that increasing the number of nodes won't improve the accuracy. We implemented an architecture with one 1024 nodes layer followed by the softmax layer. This model was again trained for 50 epochs and the best model was used for testing the accuracy which came out to be **94.33%**. This indicated that increasing the number of nodes in a single layer will not help out with the improvement in accuracy. So, we tried by adding 2 layers of 512 nodes each and a 3 node softmax layer for predictions. All the internal layers had a dropout of 0.25 and a regularizer to avoid overfitting. This model was trained for 50 epochs and the best weights gave a testing accuracy of **93.33%**. In the next model, the dropout was increased to 0.3 which might help to avoid the overfitting. But, there was no significant improvement in the model performance.

```

Epoch 0044: val_loss did not improve from 0.08582
Epoch 45/50
3679/3679 [=====] - 42s 11ms/step - loss: 0.0612 - accuracy: 0.9935 - val_loss: 0.0901 - val_accuracy: 0.9944

Epoch 0045: val_loss did not improve from 0.08582
Epoch 46/50
3679/3679 [=====] - 42s 11ms/step - loss: 0.0523 - accuracy: 0.9940 - val_loss: 0.1698 - val_accuracy: 0.9667

Epoch 0046: val_loss did not improve from 0.08582
Epoch 47/50
3679/3679 [=====] - 42s 11ms/step - loss: 0.0662 - accuracy: 0.9908 - val_loss: 0.2487 - val_accuracy: 0.9667

Epoch 0047: val_loss did not improve from 0.08582
Epoch 48/50
3679/3679 [=====] - 42s 11ms/step - loss: 0.0603 - accuracy: 0.9948 - val_loss: 0.2114 - val_accuracy: 0.9667

Epoch 0048: val_loss did not improve from 0.08582
Epoch 49/50
3679/3679 [=====] - 42s 11ms/step - loss: 0.0554 - accuracy: 0.9943 - val_loss: 0.4461 - val_accuracy: 0.9389

Epoch 0049: val_loss did not improve from 0.08582
Epoch 50/50
3679/3679 [=====] - 42s 11ms/step - loss: 0.0553 - accuracy: 0.9935 - val_loss: 0.0639 - val_accuracy: 0.9944

Epoch 0050: val_loss improved from 0.08582 to 0.06395, saving model to /model/weightsvgg.h5
done

```

*Fig 6.4.2.2: Figure showing the VGG-19 model's training*

Finally, we implemented the model which had 1 fully connected layer with 256 nodes followed by a dropout of 0.3 and a 3 node softmax layer. This model was trained with an RMSprop optimizer with a learning rate of 0.0001 and l2 regularizer with lambda value 0.01. The model had the loss function as categorical cross entropy[Figure 6.4.2.1]. This model was trained for **50 epochs** and the best model was chosen based on the minimum value of loss function. This model gave a testing accuracy of **97.5%** [Figure 6.4.2.3].

```

[ ] from sklearn.metrics import accuracy_score
    reconstructed_vgg = load_model('/model/weightsvgg.h5')
    predictions_vgg = reconstructed_vgg.predict_classes(test_imgs[0])
    print(accuracy_score(df_test.label, predictions_vgg))

```

```

[ ] 0.975

```

*Fig 6.4.2.3: Figure showing the VGG-19 model's training*

This model is the best model that we could train with the given training set and tunable parameters.



## 7. Results

After implementing several models to classify the COVID-19 radiography dataset into the three classes, it was seen that the **VGG-19** model performed the best. The model achieved an accuracy of **97.5%** on the test data and **99.4%** on the validation data. The ResNet-50 model performed the second best, achieving an accuracy of 94.16% on the test data and 95% on the validation data.

The performance parameters of different models we implemented can be seen in the tables below.

Sr No.	Model	Accuracy (%)
1	SVM	93.33
2	MLP	92.5
3	VGG-19	97.5
4	ResNet-50	94.166

*Table 7.1 Table comparing the accuracies of different models*

Sr No.	Model	Training Accuracy(%)	Training Loss	Validation Accuracy(%)	Validation Loss	Test Accuracy(%)
1	VGG-19	99.35	0.0553	99.44	0.0639	97.5
2	ResNet-50	99.10	0.0866	95.00	0.2152	94.166

*Table 7.2 Table showing the performance parameters of the CNN architectures implemented*

## 8. Conclusion

This report presents deep Convolutional Neural Network architectures along with traditional Neural Network and Support Vector Machine models to perform the detection of COVID-19 pneumonia. The training and testing was performed for four famous machine learning and deep learning algorithms to distinguish the x-ray images of normal patients and patients suffering from viral pneumonia from COVID-19 pneumonia. The VGG-19 model outperformed the other three

models. COVID-19 has already become a big threat to the lives of the people and the healthcare system. Due to the limited capacity of the doctor's time and computer-aided-diagnosis, the model such as ours can save lives of people by early screening and proper-care. The VGG-19 model performed excellently to detect COVID-19 pneumonia, even with a relatively small dataset. The model we designed can be significantly improved by expanding the dataset and trying more CNN architectures. This model could be really useful to aid the doctors and prevent the patients from reaching a more severe stage by stopping the infection spread in the early stages.

## 9. Future Work

The highest accuracy achieved in this project was 97.5, which could be improved, compared to the margin of error that can be tolerated while using machine learning to solve medical diagnosis. Our future work involves tuning these models further to reach 99.9% accuracy that is the human performance.

Another reason for lesser accuracy is less data. The 3500 images used for training are not enough for such deep models. Hence, these models tend to overfit the data. On increasing the number of epochs, due to the repetition of the same data being, the model overfits on the data. So, we must do additional **data augmentation** to achieve higher accuracy. In the current implementation, we performed data augmentation only on the COVID-19 images, but we could expand this augmentation further to all the classes, to bring the total dataset size to at least 5000 images. This increased dataset will surely help train the models better and not lead to overfitting.

It could be beneficial to try more variants of the same model and altogether new models like **DenseNet** and **Inception**. We could also implement **custom CNN** models. The implementation of custom CNN models will help train the model from scratch on this specific problem. With data augmentation, we can have enough data to train the model in order to get the weights to match our expectations. The custom implementation will involve experimenting with the number of layers, and also the filter sizes for each convolution and pooling layers. In the end, we will be appending fully connected layers and a softmax function to make the final prediction.

**DenseNet-121** is another deep learning convolution neural network that we could implement to perform better on our data. The densenet has shown better results than the state-of-the-art models[13]. We plan to use the keras implementation of densenet-121[4]. While experimenting this, we could play around with the fully connected layers. In one approach we will be using the weights from the imagenet dataset and freeze them and only train to optimize weights of the fully connected layer. Another approach would be to initialize the network with imagenet weights but train all the weights throughout the network. We will monitor the loss function and determine the best parameters for the network based on the validation loss.

We plan to use a similar approach for the inception model. The implementation will be done using the keras implementation of **InceptionV3**[4]. The main advantage of an inception model is

that the depth of the model is increased without increasing the computational requirement[14]. Increasing the depth will help more important features that can be key elements in the classification. These different methods could definitely be implemented to expand our project and make it more suited for application in the healthcare industry.

## Bibliography

- [1] <https://www.kaggle.com/tawsifurrahman/covid19-radiography-database>
- [2] Sethy, P.K, Behera, S.K, “Detection of Coronavirus Disease (COVID-19) Based on Deep Features.”
- [3] Asmaa Abbas, Mohammed M. Abdelsamea, Mohamed Medhat Gaber, “Classification of COVID-19 in chest X-ray images using DeTraC deep convolutional neural network”
- [4] <https://keras.io/applications/>
- [5] Rumelhart, D.E., & McClelland, J.L. (1986). Parallel distributed processing (Vol. 1). Cambridge, MA: MIT Press.
- [6] Lutz Prechelt, “Early Stopping - but when?”.
- [7] M Moreira and E Fiesl, “Neural Networks with Adaptive Learning Rate and Momentum Terms”, October 1995.
- [8] Muhammad E. H. Chowdhury, Tawsifur Rahman, Amith Khandakar, Rashid Mazhar, Muhammad Abdul Kadir, Zaid Bin Mahbub, Khandakar Reajul Islam, Muhammad Salman Khan, Atif Iqbal, Nasser Al-Emadi, and Mamun Bin Ibne Reaz, “Can AI help in screening Viral and COVID-19 pneumonia?”
- [9] Shervin Minaee , Rahele Kafieh , Milan Sonka , Shakib Yazdani , Ghazaleh Jamalipour Soufi, “Deep-COVID: Predicting COVID-19 From Chest X-Ray Images Using Deep Transfer Learning”, Apr 2020
- [10] Ali Narin, Ceren Kaya, Ziyne Pamuk, “Automatic Detection of Coronavirus Disease (COVID-19) Using X-ray Images and Deep Convolutional Neural Networks”
- [11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, “Deep Residual Learning for Image Recognition”
- [12] Karen Simonyan, Andrew Zisserman , “Very Deep Convolutional Networks for Large-Scale Image Recognition.”
- [13] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, “Densely Connected Convolutional Networks.”
- [14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, “Going Deeper with Convolutions. ”

[15] ] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. Summers, "Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases," IEEE CVPR, 2017

[16] A. w. Linda wang, "COVID-Net: A Tailored Deep Convolutional Neural Network Design for Detection of COVID-19 Cases from Chest Radiography Images," 2020.

[17] S. Wang, B. Kang, J. Ma, X. Zeng, M. Xiao, J. Guo, et al., "A deep learning algorithm using CT images to screen for Corona Virus Disease (COVID19)," medRxiv, 2020

[18] <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>

[19] [https://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPClassifier.html](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)