**CS 4662 Final Project Report**

**Group members:**

Deqing Liang, Yingpeng Ma, Abulaiti Akeli

**Members’ responsibility**:

Deqing Liang: Training and Testing Random Forest Algorithm.

Yingpeng Ma: Training and Testing ANN and CNN.

Abulaiti Akeli: Training and Testing CNN and SVM, data pre-processing.

All members are responsible for the project report writing and the making of presentation slides.

**Project name:**

Covid-19 Image Dataset 3 Way Classification

**Project description and details:**

This is a project from Kaggle website ([Covid-19 Image Dataset | Kaggle](https://www.kaggle.com/pranavraikokte/covid19-image-dataset)). To help the medical and research community and encourage them to contribute extensively, this project aims to build a 3 way classification model to classify X-ray images of COVID-19, Viral Pneumonia and normal chest. The Covid 19 Image dataset is also provided by the publisher, further detail about the dataset will be covered in the following section. Basically, our task is to use different types of machine learning or deep learning algorithms to train and test out which chest x-ray images are COVID-19, normal, or Viral Pneumonia, with limited data provided by the publisher. In short, we need to build classification models with 3 different classes.

**Project goal:**

The original goal of this project which is posted on the Kaggle website is to help Deep Learning and AI Enthusiasts to contribute to improving COVID-19 detection using just Chest X-rays, which would help the medical and research community and encourage them to contribute extensively on COVID-19 studies. Besides this very ambitious goal, our more realistic goal is to try our best to utilize all we have learned in this class to build and improve the machine learning and deep learning models to reach the highest accuracy we can possibly reach within our limits, and gain some precious experience and knowledge throughout this project.

**Details about data:**

All of the images in this dataset was provided by The University of Montreal.The data set contains 317 cleaned lungs’ X-ray images in total. The size of the data set is relatively low to train a very accurate model. The dataset is already pre-divided into training and testing directories, therefore no further data split would be needed for our models. Each of those directories branches into three different classes, such as Covid-19, Viral Pneumonia, and Normal. The training directory contains 251 images in total, 111 for COVID-19, 70 for normal and 70 for Viral Pneumonia. The testing set on the other hand also includes 66 images in total, 26 for COVID-19 and 20 for Normal and Viral Pneumonia each. The images’ format and size are not uniformed, which are important factors we need to take into consideration when we import them.

**Developed methods, algorithms, tools:**

**Methods:**

**Image Resize and Reshape:** We used image resize method to reduce the images size for better processing speed, and we also used image reshape method to turn the images into the feature set of pixels and change the color channel .

**Random Shuffle:** We used this method to make sure the images of the same categories are not appearing consecutively, thus improving the accuracy of our model and avoiding false positive training results.

**Scale Normalization:** We used this method to normalize the pixel values of our feature set between 0 and 1.

**One Hot Encoding:** A method of manipulating categorical data to be more readable for machine/deep learning algorithms by creating a new field for each nominal category of a categorical variable. In our case, we turned our 3 prediction classes from “Covid”, “Normal”, “Viral Pneumonia” into 0, 1, 2.

**Grid search:** A tuning technique that attempts to compute the optimum values of hyperparameters. It is an exhaustive search that is performed on the specific parameter values of a model. We used this technique to find the best epochs and batch size for our sequential models, best neuron size for the ANN model, and best ‘C’ parameter for our SVC model.

**10 fold cross validation:** An algorithm of repeating the splitting process of our data multiple times, in our case 10 times, in order to utilize each data sample in the testing set one time and in the training set ‘k-1’ times; culminating the prediction accuracy of each split and then averaging the result.

**ROC Curve and AUC**: ROC is the plot between TPR(True Positive Rate) and FPR(False Positive Rate) across all possible thresholds and AUC is the entire area beneath this ROC curve. We used this method to evaluate and visualize the performance of our MLP-ANN, SVM and Random Forest models.

**PCA:** Dimensionality reduction method to remove redundant and unwanted parts from our feature set to speed up the training process and accuracy of our model. We used it on ANN and SVM models.

**Confusion Matrix**: It is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one, thus we used it for SVM and Random Forest models. Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class, or vice versa .

**Algorithms:**

**CNN (Convolutional Neural Network):** Type of neural network model which allows us to extract higher representations for the image content. Unlike the classical image recognition where you define the image features yourself, CNN takes the image’s raw pixel data, trains the model, then extracts the features automatically for better classification.

**ANN (Artificial Neural Network):** It’s a computational model which is based on structures and functions of biological neural networks. We implemented 2 ANN models with keras classifier and MLP classifier.

**SVM (Support Vector Machine):** A supervised learning model with associated learning algorithms that analyze data for classification and regression analysis. SVM maps training examples to points in space so as to maximize the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

**Random Forest:** It is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees.

**The details of the implementation of these models and algorithms will be covered in the following section.**

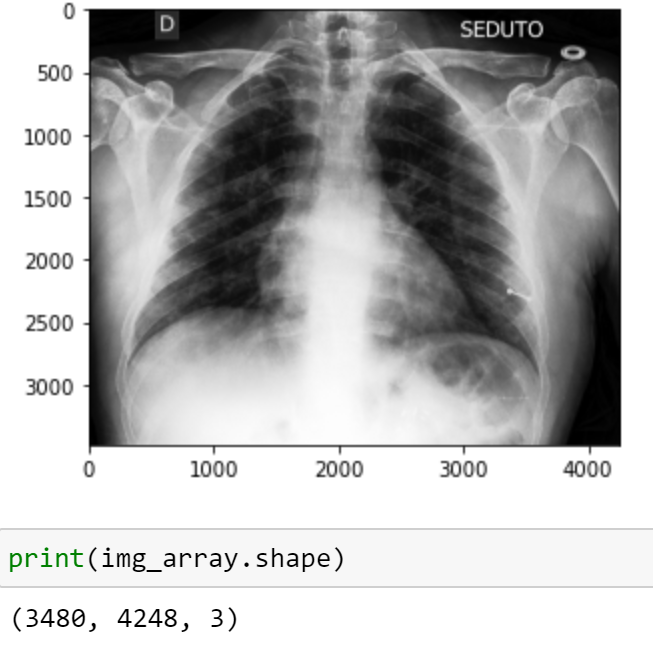
**Tools:**

1. Jupyter Notebook
2. Numpy
3. Open CV
4. OS
5. Scikit-learn
6. Tensorflow
7. Matplotlib

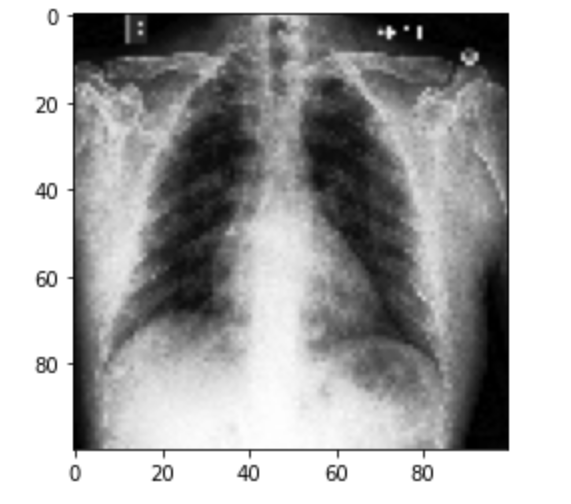
**Developed codes and results:**

**Data Pre-Processing:**

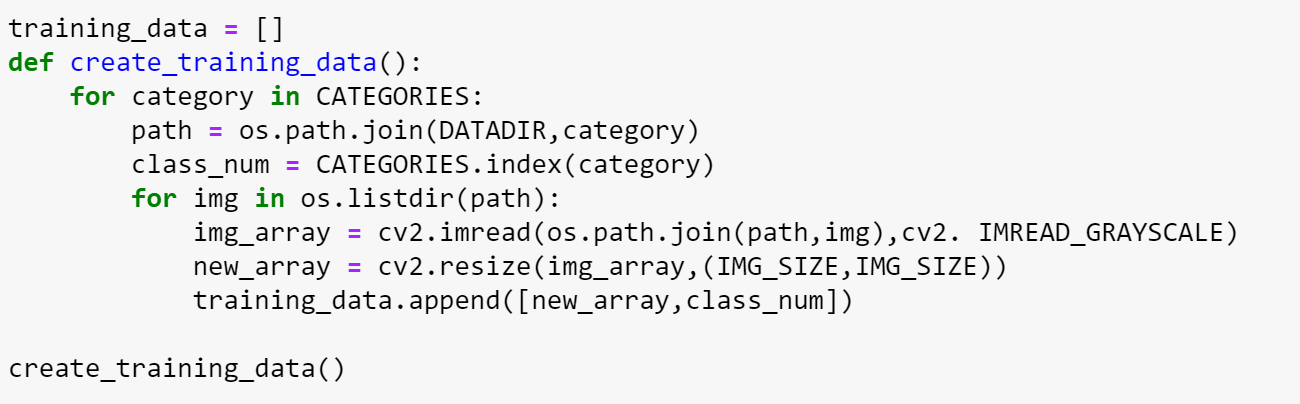
The first thing we want to do for all this kind of project is check out the images from the dataset:

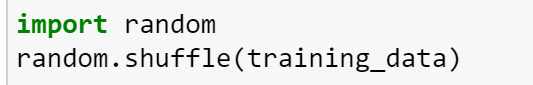


As shown above, these images are very big. Our computers definitely can’t handle this size for the training, so we have to reduce the images’ sizes. Also, notice here that despite the X-Ray images being all black and white, these images are still in the RGB format, so we still need to convert them into grayscale images to reduce the models’ complexity and training time. We also plot some other images and notice their sizes and shapes are slightly different, we need to uniform the size and the shape of the images as well.

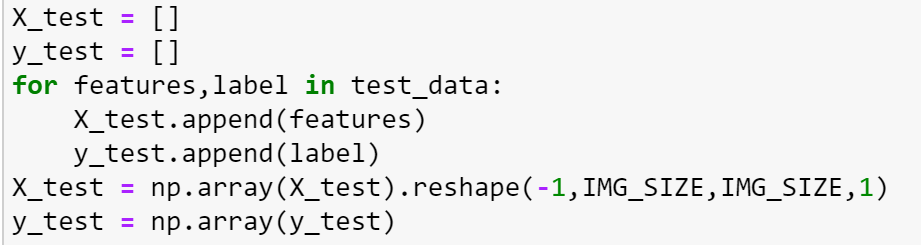


Here is the image after we reduce its size to 100 x 100. It’s a huge cut in the size and the image is looking quite fuzzy to the eyes. However, we were able to get pretty decent prediction results for our models using this image size. We tried to raise the size all the way up to 300 x 300, but the prediction results didn’t really improve. In the end, we stick with the size of 100 x 100.

****

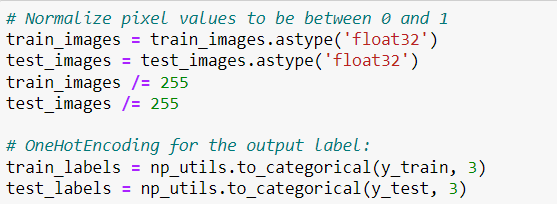
****

Above is how we import the images and create the dataset. Since it loops through every folder under the train/test directories one by one, all the images within the same categories will be following each other consecutively in the dataset. We want to avoid that and implement the random shuffle method to mix up the images in the dataset.



Above is the final tweak for our dataset, we divide the features and labels list and turn them into np arrays. We also reshape the images into (100, 100, 1) shape. This is only temporary at this point since different models will require different shapes of feature sets.

**Convolutional Neural Network Algorithm:**

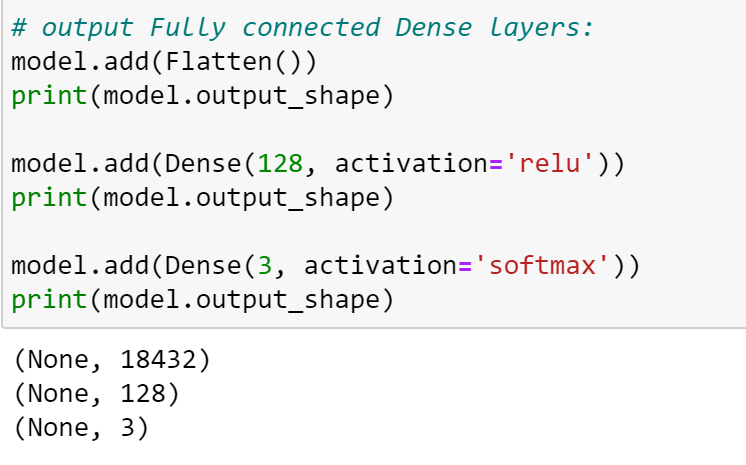
****

As we mentioned before, we need to change the shape and normalize the values of our features and labels set according to the model we are using. In this case, we first normalize the pixel values to be between 0 and 1, and then use the OneHotEncoding for the output labels.

We learned from the class that CNN is a very strong deep learning algorithm for the images. It has lots of different designs and implementations, therefore tremendous potential. This was the first algorithm we came up with and we believed it would provide the best results. However, we realized later that CNN is not always the no.1 choice for everything, especially if we have a small size of data. In our case, we have less than 300 images to train, which is definitely considered as a small data size. We started from a very basic simple CNN model and got 0.85-0.92 accuracy which we consider pretty decent already. Unfortunately, the good news stopped there and everything beyond ‘basic’ didn’t give us any noticeable improvement, it feels like it already reached its limit. We think that is probably where the disadvantage of the CNN model dealing with small datasets is exposed. We also tried to implement data augmentation, but it even reduced the accuracy. One thing we noticed is that while having data augmentation of 1 or 2 operations reduces the accuracy, the accuracy starts to go up as we increase the number of operations more and more. Maybe in the end at some point we could get a better result with data augmentation but we think the cost wasn't worth it so we decided to give up on that one.

Blew is the final structure of our CNN model. Again, the accuracy difference we got from different structures was not obvious at all. What made things even worse was that we got different results even from the same structure each time when we ran it, so it was really hard for us to tell whether we were actually making improvements or not. Anyways, we decided to use 4 convolutional layers with different filter sizes, each with 2 x 2 maxpooling. We ditched drop out layers as we noticed they reduced the accuracy, probably because our CNN model is not very complex so it was not needed. As for the 3 fully connected layers, we got 1 flatten layer, 1 dense layer with value 128, and the final dense layer with 3 categories for classification.

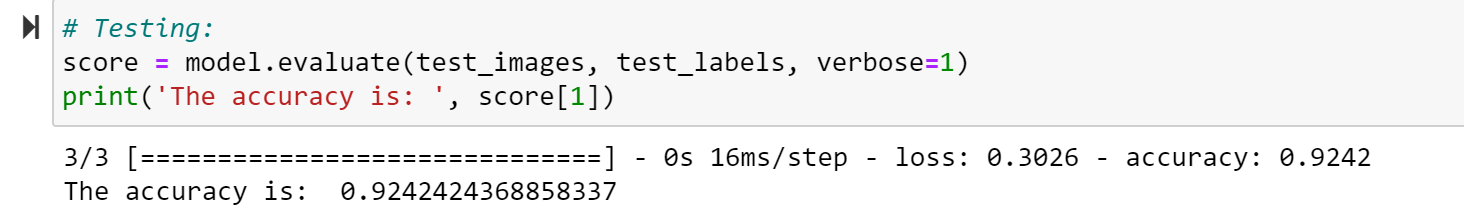
****

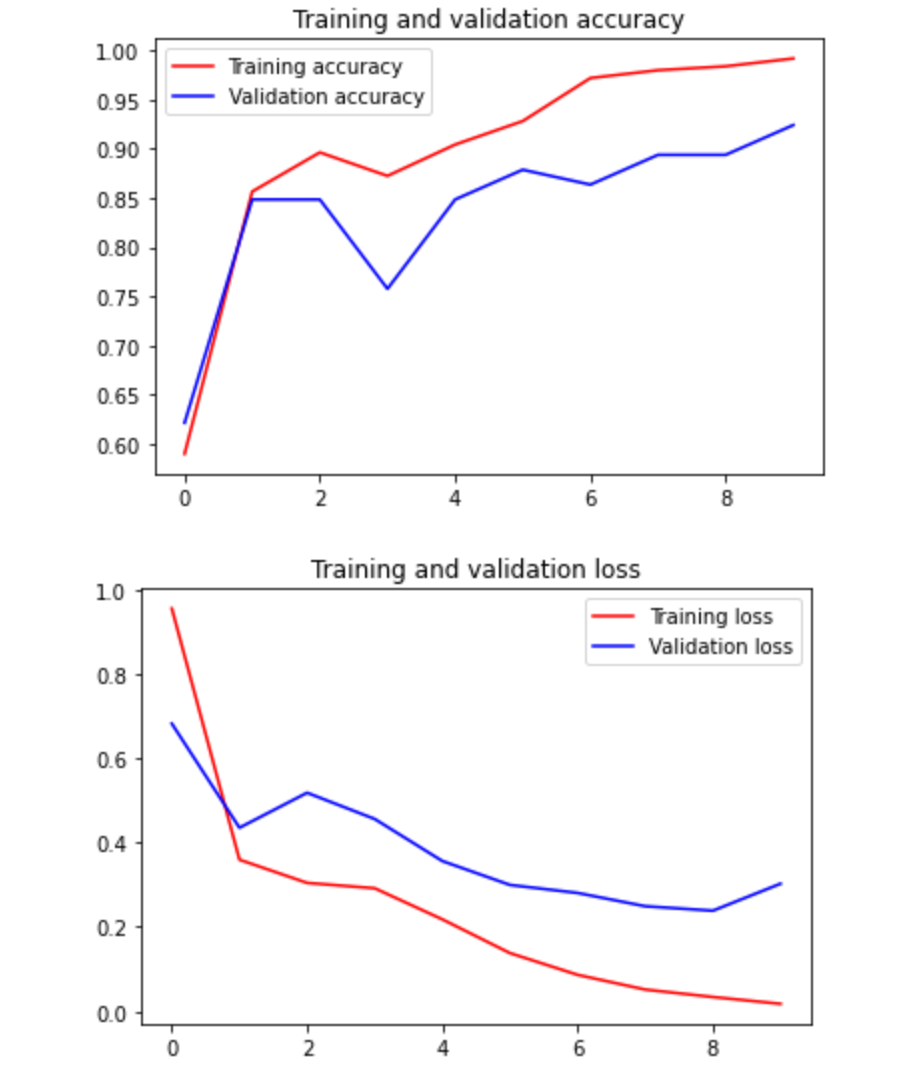
****

When we were deciding on which batch size and epoch to use for our CNN model, we did try the grid-search method but for some reason it was taking an abnormally long time even only searching for 3 batch sizes and 3 epochs. It was way quicker to just manually try out different batch sizes and epochs. After several tries we decided to use a batch size of 30 and epoch of 10.



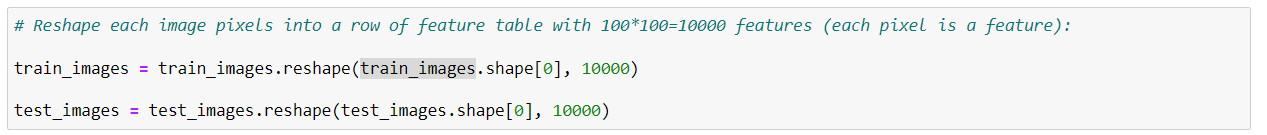
**Result:**

****

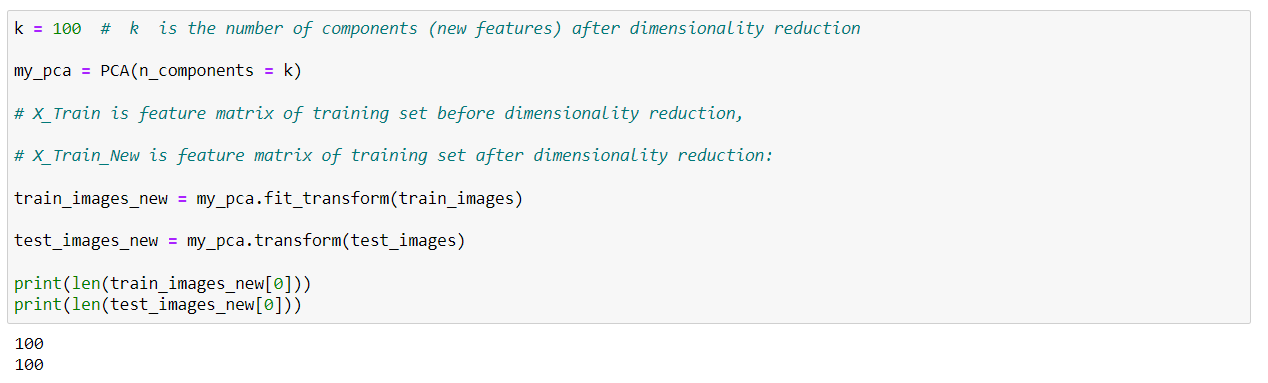
****

**Artificial Neural Network:**

For the ANN, we implemented it by both sklearn and tensorflow. Before training the model, we reshape each image pixels into a row of feature table with 100\*100 = 10000 features as follows:



Then, we used PCA to do the dimensionality reduction in order to get a better performance and train the dataset faster, for the parameter k, which is the number of components after PCA, we put 100:

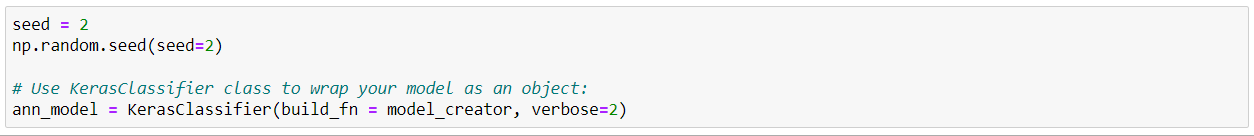


*1) Tensorflow library*

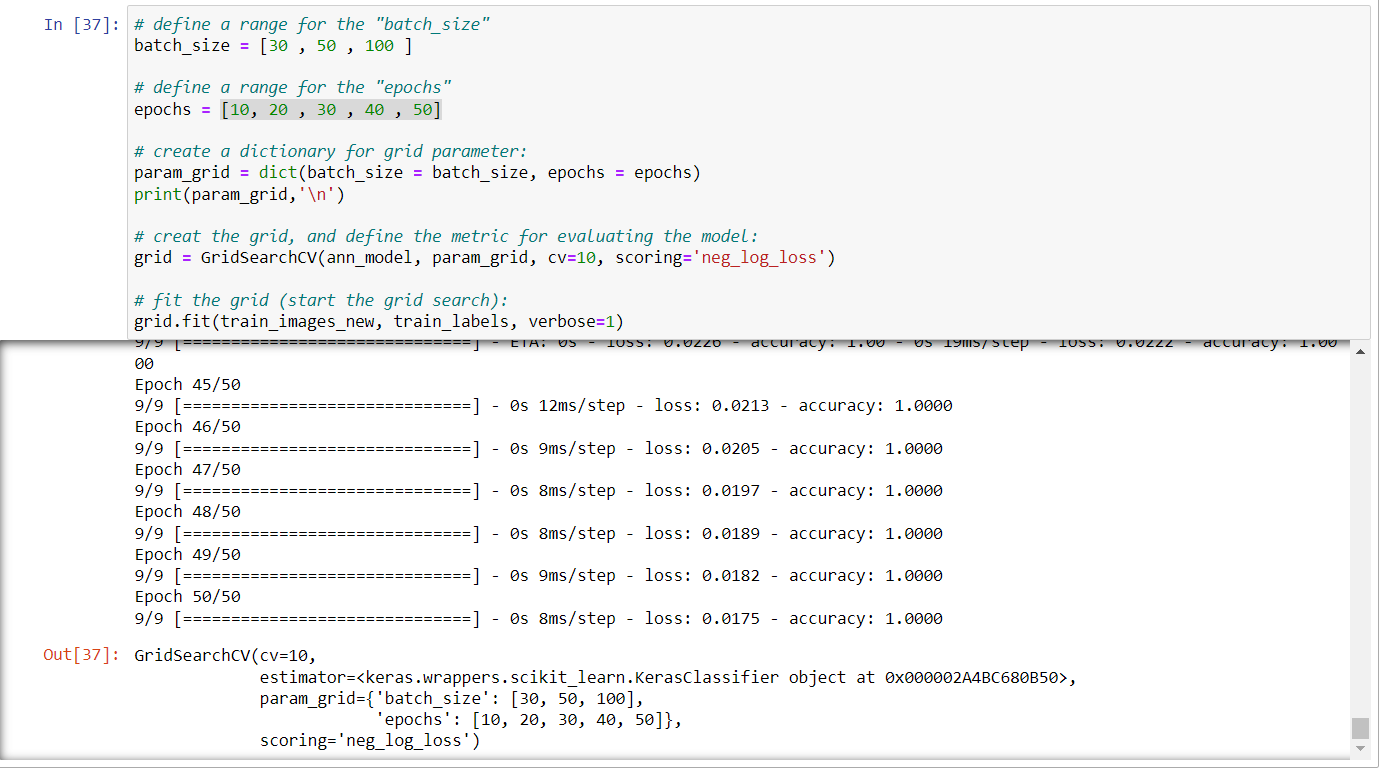
First, we define the model\_creator function to build our ANN model just like what we did in the lab. In this function, we configure 100 input features, which is the same size after dimensionality reduction(PCA), 100 hidden\_neurons and only output 3 features:



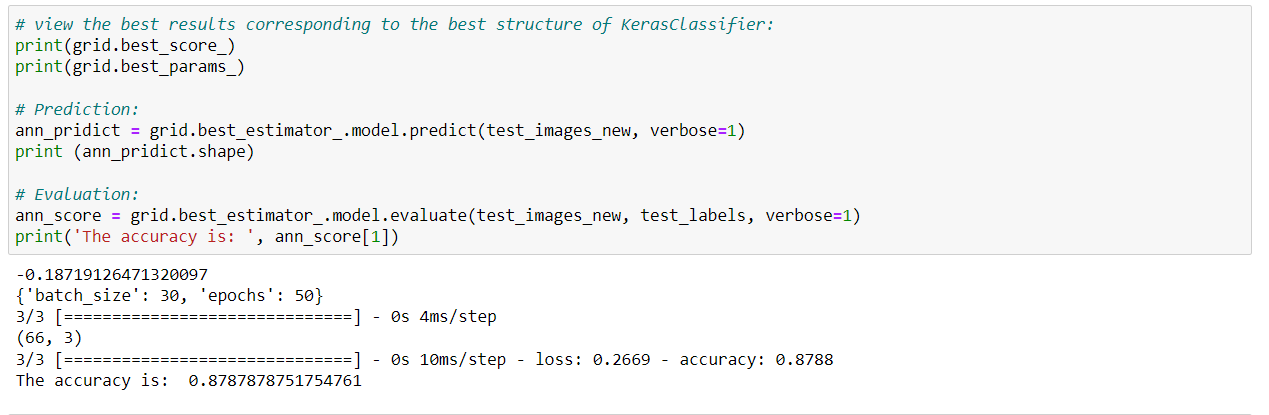
Next, we fix the random state for reproducibility and use KerasClassifier class to wrap our model as an object for gridsearch using:

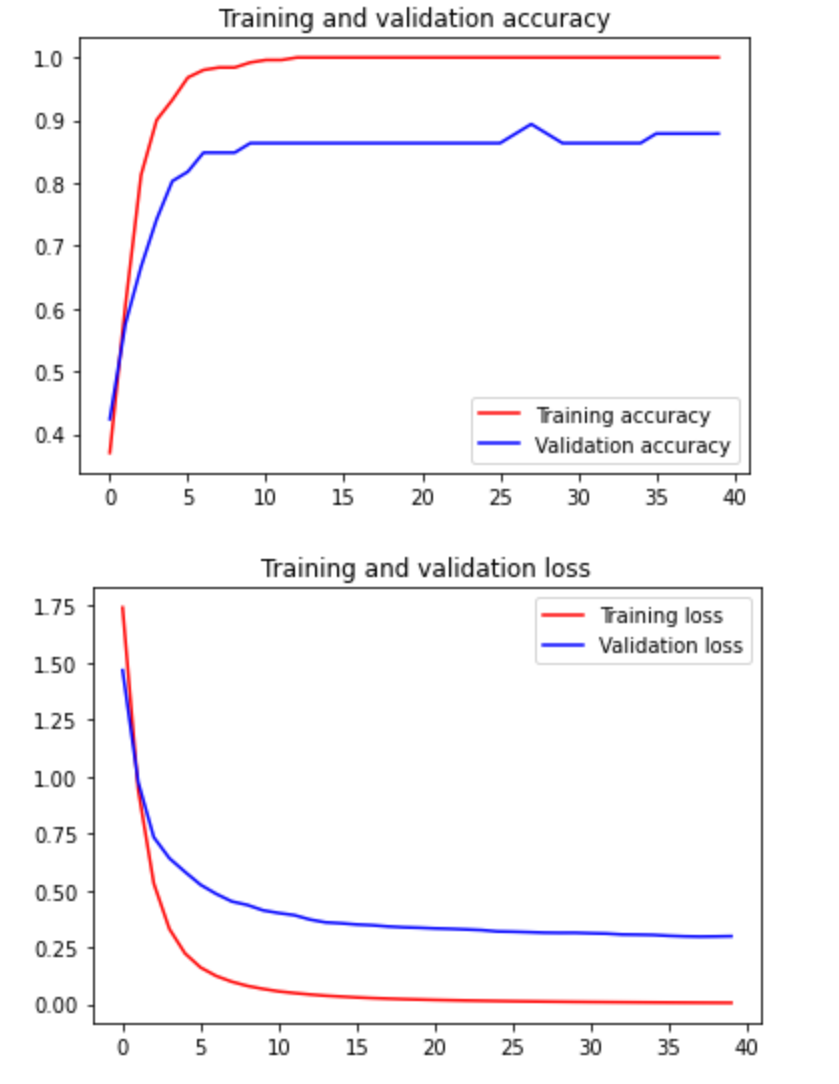


To find the best batch size and epochs, we use gridsearch to look for them. For the batch size, we set the range list [30 , 50 , 100 ] and for the epochs, we set the range list [10, 20 , 30 , 40 , 50], the code of gridsearch and result are shown below:



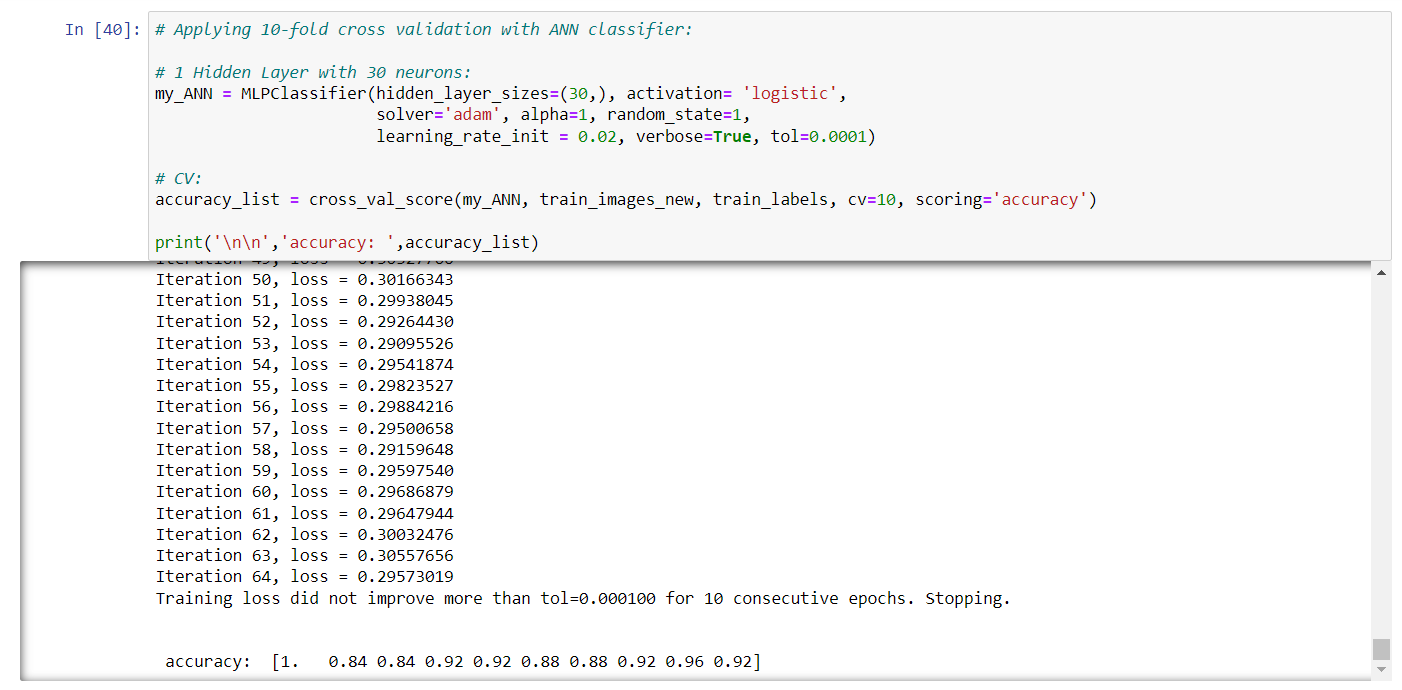
Finally, we found the best parameter is batch size: 30, epochs: 50, and the accuracy of this algorithm is around 0.8788:





*2) Scikit-Learn library*

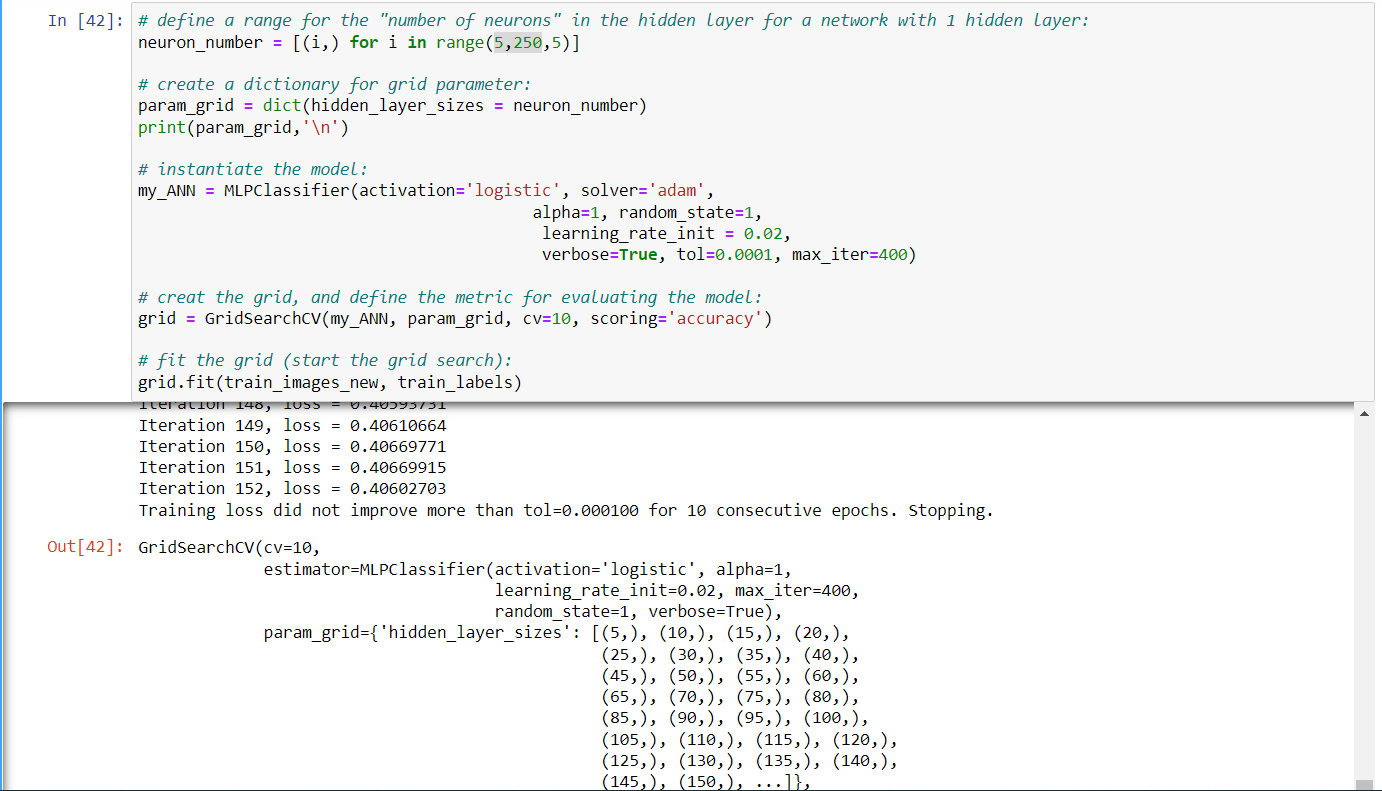
First, we define the MLPClassfier to build our ANN model with the parameter 30 hidden layer, logistic activation function, random\_state = 1, learning\_rate\_init = 0.02 and the Tolerance for the optimization = 0.0001. Moreover, we apply 10-fold cross validation with this classifier as shown below:



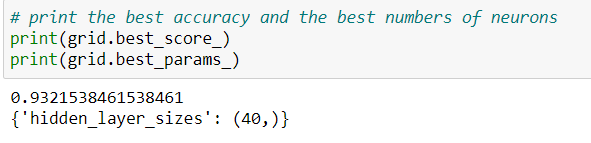
With these 10 -fold cross validation accuracy, we compute the average of them and get the final accuracy 0.908 as follows:

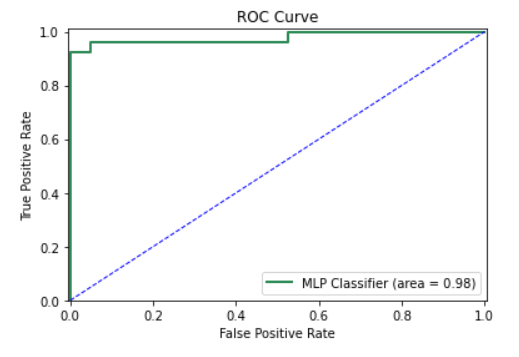


Just like what we did in Tensorflow above, in order to find the best neuron\_number, we use gridsearch to look for it. We set the range from 5-250 and make each value’s interval to 5, the code of gridsearch and result are shown below:



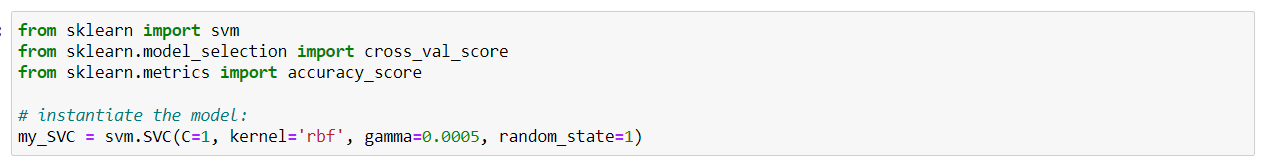
Finally, we found the best parameter is hidden\_layer\_sizes: 40, and the accuracy of this algorithm is around 0.9321:



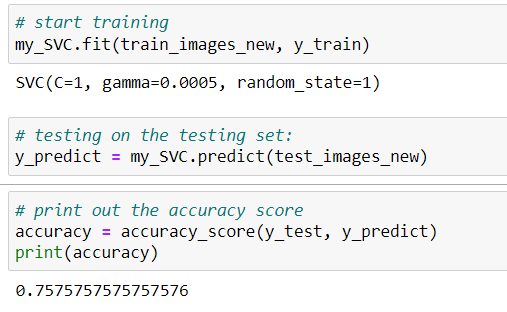


**Support Vector Machine Algorithm:**

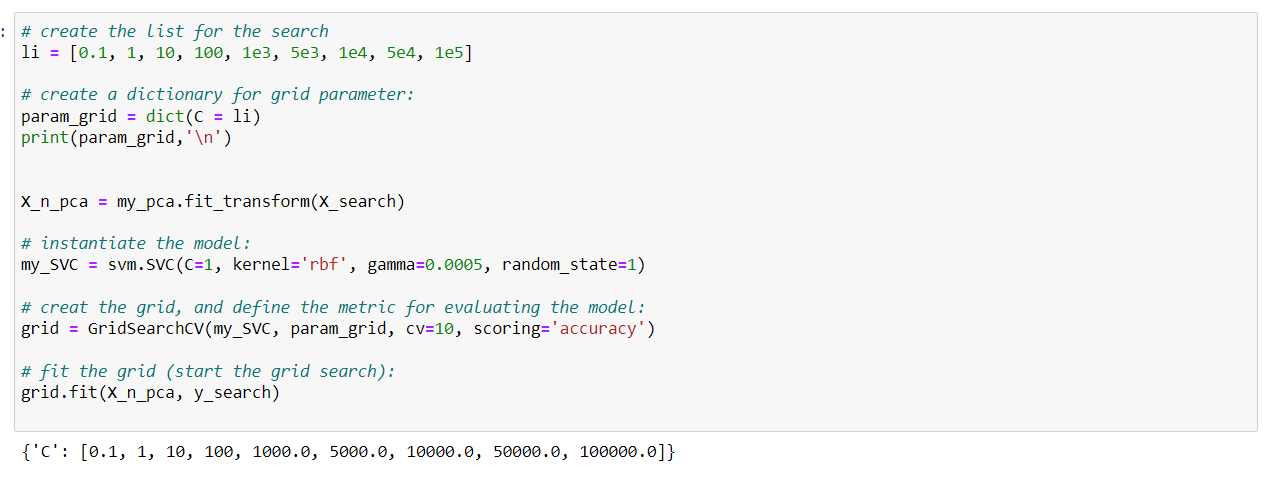
To build our SVM model, we import the following library shown in the screenshot and set the C=1, gamma=0.0005, random\_state=1 just for testing:

****

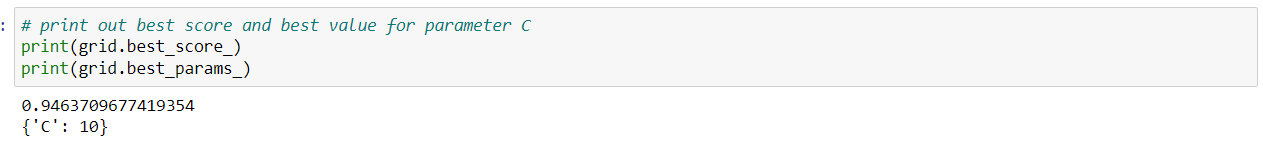
Moreover, we use the training dataset after PCA that we describe above in ANN to train and test our SVM model, however, the result is not so ideal, the accuracy is only 0.7575, which we think is too low as shown below:

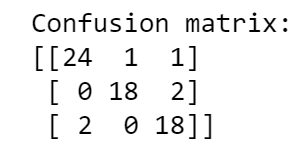


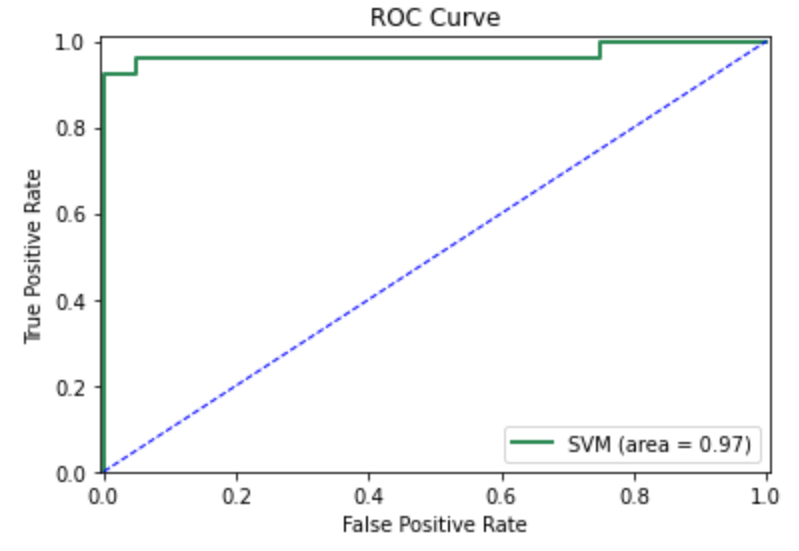
Therefore, we try to use gridsearch again to find the best parameter of C, we define the range list [0.1, 1, 10, 100, 1e3, 5e3, 1e4, 5e4, 1e5], and keep other parameter the same as the first one we implement it again as shown below:



Finally, we get an accuray 0.9463 by using the parameter C=10, which we think it’s good enough

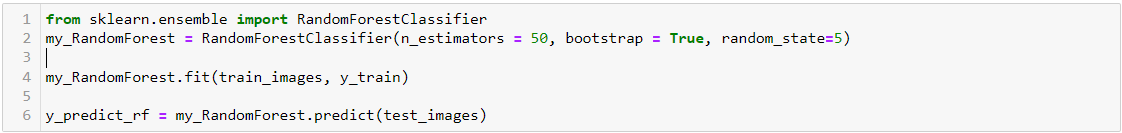




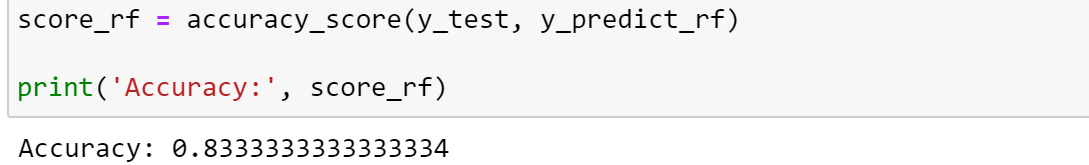


**Random Forest Algorithm :**

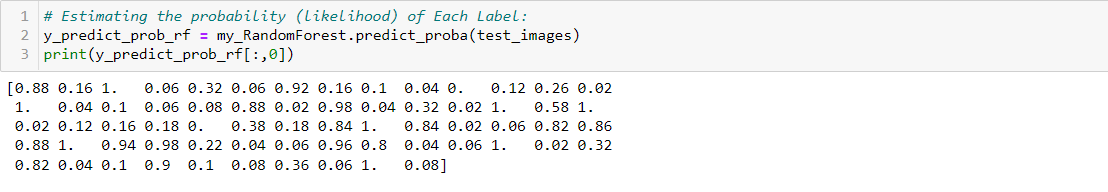
//Import RandomForest Classifier as usual

//In the beginning, the provider had already had train and test image folders split for us. Therefore, we don’t need to use the split and train method to split part of the data for testing purpose, and we simply reshaped the image into the size of 100 \* 100 and added them to corresponding datasets.

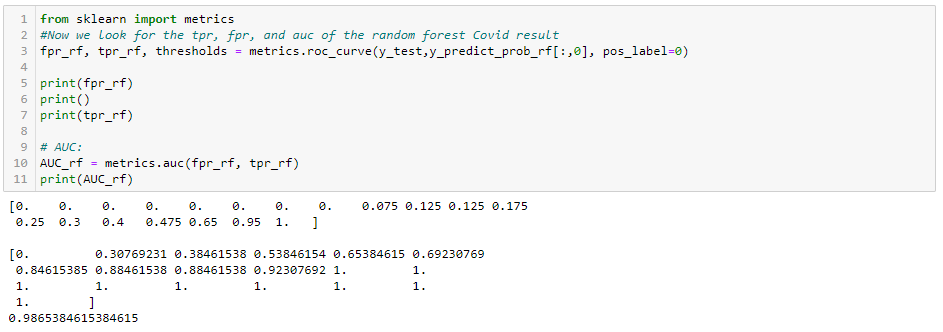
//We train the data and test predict the outcome accuracy, which equal to 0.83333333334

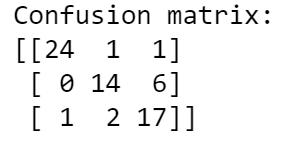


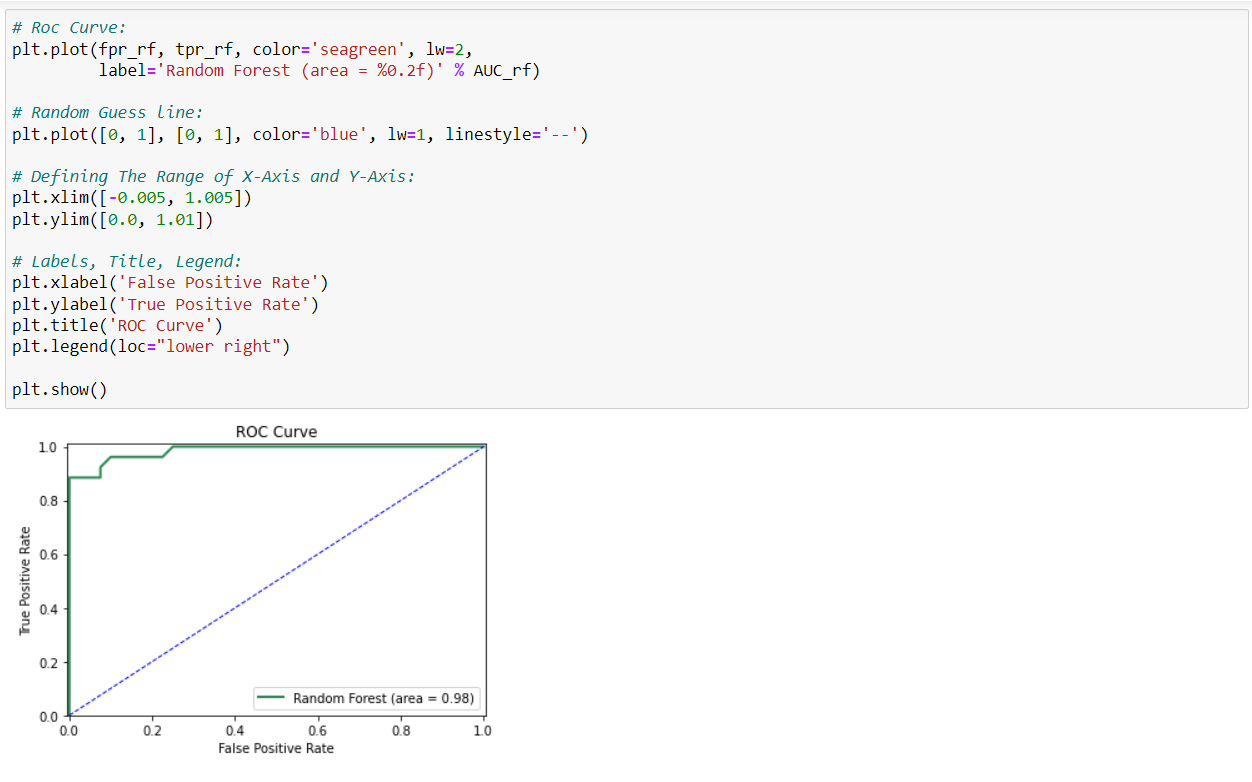
//Then, we predict the probability of each label



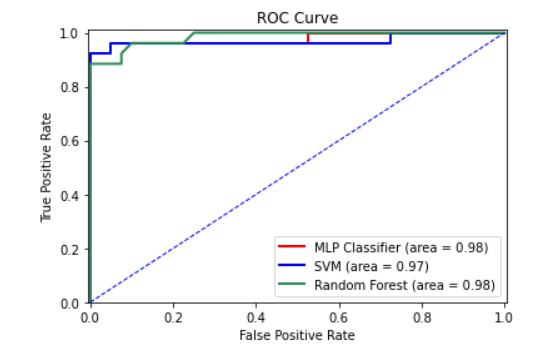
// Then, we look for the correct positive result, incorrect positive result, and the area under curve of the random forest Covid result.







**ROC of Three Algorithms:**



**Conclusion & Future Work:**

Based on the accuracy outcome, we conclude that Support Vector Machine delivers the best results for our specific classification task, which is 0.9463.

* The algorithms we use provide us with great visualization and contributions to improving COVID-19 detection.
* We would like to explore other methods/algorithms in the future like:
* K Nearest Neighbor
* Recurrent Neural Networks (RNNs)
* Generative Adversarial Networks (GANs)