ISS Project (Graduate Certificate in Pattern Recognition Systems)

Intelligent Mask Detection Platform

Detective Mask

Hong Xiaohui (SXXXX943D)

Anita Koo Shi Qi (SXXXX480B)

Sanjeven Ramakrishnan (SXXXX938E)

National University of Singapore | Institute of Systems Science

https://github.com/xiaohuihong/PRS-PM-2021-09-15-GRP-3Musketeers-DetectiveMask

Table of Content

Abstract: With the recent spike in community cases of covid-19, we must remain vigilant although the majority of our population are vaccinated. As Singapore slowly opens its borders and embraces the new normal, we need to make arrangements to transit smoothly. Once such arrangement would be to redeploy staff from patrolling common areas to ensure people are wearing masks. This project proposes a solution to this problem by using a machine learning model to determine if the subject is wearing a mask and to provide an alert to authorities if there is a breach.

1. Business Case

Recent news reports have shown that there has been a significant increase in local community covid-19 cases. This has led to covid-19 helplines being overwhelmed due to lack of operators and for resources to be used up at a significantly higher rate. There is also news of Singapore opening their borders via the vaccinated travel lines as Singapore moves towards an endemic state. As we move towards this 'new normal' we would need to streamline our processes for us to be sustainable as a country and to allocate resources efficiently. One such avenue is the jobs carried out by officials who patrol to ensure that all patrons are always wearing their masks.

We propose that this can be handled using a web application that is able to provide real-time mask detection in public places and to provide a form of real-time alert to authorities to enforce the law when there is a breach.

2. System Models

For this application, we evaluated 3 models (KNN, SVM and CNN) and the metrics we used to evaluate are accuracy, F1-score and loss.

2.1. KNN Model

You can find the details of this model in **../source/knn_train_model_v1.ipynb**

A total of four iterations were carried out for the KNN model.

For the first iteration, neither hyperparameter tuning nor dimension reduction was used. For the second iteration, dimension reduction was used - PCA (0.9). GridSearchCV was used for Hyperparameter tuning with the following parameters of :

- n_neighbours : 5, 7, 9 and 11
- p : 2, 4, 6, 8, 10, 20 and 50
- metric: euclidean, manhattan and minkowski

KNN Hyperparameter Tuning with GridSearchCV

```
# New KNN instance model
knn2 = KNeighborsClassifier()

# Dictionary of parameter values we are testing performance for
param_grid = {'n_neighbors': [5, 7, 9, 11], 'p': [2, 4, 6, 8, 10, 20, 50], 'metric': ['euclidean', 'manhattan', 'minkowski']}

# Test all parameter combinations in param_grid
knn_gscv = GridSearchCV(knn2, param_grid, scoring='f1_micro', cv=5, verbose=3, n_jobs=-1)
```

```
# Fit model to data
knn_gscv.fit(trDat, trLbl)
```

```
Fitting 5 folds for each of 84 candidates, totalling 420 fits
```

```
# Get GridSearchCV's top performing parameters and score

# Best Params: n_neighbors=5, distance=manhattan, p=2
best_params_ = knn_gscv.best_params_

# Best Score 82.7%
knn_gscv.best_score
```

Figure 1 Code snippet for getting the best parameters

After getting the best parameters to use for the KNN model, a third iteration was carried out using just the hyperparameters. For the fourth iteration, the hyperparameter was used with dimension reduction. The accuracy results of the four iterations are as follows: 81.5%, 83.9%, 80.9% and 78.8%.

```
KNN Precision: 0.815
KNN Recall: 0.815
KNN F1 Score: 0.815

No Hyperparameter Tuning Classification Report
              precision    recall  f1-score   support

           0       0.90      0.70      0.79       745
           1       0.76      0.92      0.84       766

   micro avg       0.82      0.82      0.82      1511
   macro avg       0.83      0.81      0.81      1511
weighted avg       0.83      0.82      0.81      1511
 samples avg       0.82      0.82      0.82      1511
```

Figure 2 First iteration confusion matrix

```
Accuracy Train PCA: 0.8391793514228988
KNN Recall: 0.839
KNN F1 Score: 0.839
Wall time: 272 ms
```

Figure 3 Second iteration scores

```
Accuracy Train: 1.000
Accuracy Test: 0.809
KNN Recall: 0.809
KNN F1 Score: 0.809

Classification Report
              precision    recall  f1-score   support

           0       0.84      0.76      0.80       745
           1       0.78      0.86      0.82       766

   micro avg       0.81      0.81      0.81      1511
   macro avg       0.81      0.81      0.81      1511
weighted avg       0.81      0.81      0.81      1511
 samples avg       0.81      0.81      0.81      1511
```

Figure 4 Third iteration confusion matrix

```
Accuracy Train PCA: 0.7875579086697552
KNN Recall: 0.788
KNN F1 Score: 0.788
Wall time: 2.71 s
```

Figure 5 Fourth iteration score

2.2. SVM Model

You can find the details of this model in **../source/svm_rbf.ipynb and svm_poly.ipynb**

For SVM, we used scikit-learn's SVM module. We carried out 8 iterations with different combinations of C and gamma. 4 were with radial basis function (rbf) kernel and 4 were with polynomial kernel for our model. The gamma values used were 0.1 and 1. The C values used were 1 and 10.

```
gamma1, gamma2 = 0.1, 1
C1, C2 = 1, 10
hyperparams = (gamma1, C1), (gamma1, C2), (gamma2, C1), (gamma2, C2)

scaler = StandardScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)

svm_clfs = []
for gamma, C in hyperparams:
    svm_clf=SVC(kernel="rbf",gamma=gamma, C=C)
    svm_clf.fit(x_train,y_train)
    svm_clfs.append(svm_clf)
```

Figure 6 Code snippet for choosing parameters and rbf kernel type

```
r= 0.1 C= 1
Accuracy= 0.5519523494374586
[[ 68 677]
 [  0 766]]
              precision    recall  f1-score   support

           0       1.00      0.09      0.17       745
           1       0.53      1.00      0.69       766

    accuracy                           0.55      1511
   macro avg       0.77      0.55      0.43      1511
weighted avg       0.76      0.55      0.43      1511

r= 0.1 C= 10
Accuracy= 0.5519523494374586
[[ 68 677]
 [  0 766]]
              precision    recall  f1-score   support

           0       1.00      0.09      0.17       745
           1       0.53      1.00      0.69       766

    accuracy                           0.55      1511
   macro avg       0.77      0.55      0.43      1511
weighted avg       0.76      0.55      0.43      1511
```

```
r= 1 C= 1
Accuracy= 0.5506287227001986
[[ 66 679]
 [  0 766]]
              precision    recall  f1-score   support

           0       1.00      0.09      0.16       745
           1       0.53      1.00      0.69       766

    accuracy                           0.55      1511
   macro avg       0.77      0.54      0.43      1511
weighted avg       0.76      0.55      0.43      1511

r= 1 C= 10
Accuracy= 0.5506287227001986
[[ 66 679]
 [  0 766]]
              precision    recall  f1-score   support

           0       1.00      0.09      0.16       745
           1       0.53      1.00      0.69       766

    accuracy                           0.55      1511
   macro avg       0.77      0.54      0.43      1511
weighted avg       0.76      0.55      0.43      1511
```

Figure 7 confusion matrix for each iteration

The accuracy from all 4 iterations using rbf are low and similar. The accuracy of the iterations are 55.19%, 55.19%, 55% and 55%.

```
gamma1, gamma2 = 0.1, 1
C1, C2 = 1, 10
hyperparams = (gamma1, C1), (gamma1, C2), (gamma2, C1), (gamma2, C2)

scaler = StandardScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)

svm_clfs = []
for gamma, C in hyperparams:
    svm_clf=SVC(kernel="poly",gamma=gamma, C=C)
    svm_clf.fit(x_train,y_train)
    svm_clfs.append(svm_clf)
```

Figure 8 Code snippet for choosing parameters and poly kernel type

```
r= 0.1 C= 1
Accuracy= 0.885506287227002
[[631 114]
 [ 59 707]]
              precision    recall  f1-score   support

           0       0.91      0.85      0.88       745
           1       0.86      0.92      0.89       766

    accuracy                           0.89      1511
   macro avg       0.89      0.88      0.89      1511
weighted avg       0.89      0.89      0.89      1511

r= 0.1 C= 10
Accuracy= 0.885506287227002
[[631 114]
 [ 59 707]]
              precision    recall  f1-score   support

           0       0.91      0.85      0.88       745
           1       0.86      0.92      0.89       766

    accuracy                           0.89      1511
   macro avg       0.89      0.88      0.89      1511
weighted avg       0.89      0.89      0.89      1511
```

```
r= 1 C= 1
Accuracy= 0.885506287227002
[[631 114]
 [ 59 707]]
              precision    recall  f1-score   support

           0       0.91      0.85      0.88       745
           1       0.86      0.92      0.89       766

    accuracy                           0.89      1511
   macro avg       0.89      0.88      0.89      1511
weighted avg       0.89      0.89      0.89      1511

r= 1 C= 10
Accuracy= 0.885506287227002
[[631 114]
 [ 59 707]]
              precision    recall  f1-score   support

           0       0.91      0.85      0.88       745
           1       0.86      0.92      0.89       766

    accuracy                           0.89      1511
   macro avg       0.89      0.88      0.89      1511
weighted avg       0.89      0.89      0.89      1511
```

Figure 9 confusion matrix for each iteration

The accuracies from all 4 iterations using poly are similar. The accuracy of the iterations is 88.55%.

2.3. CNN Model

2.3.1. Overview

Other than the two supervised machine learning models, KNN and SVM, we chose a deep learning model which is CNN to make the image classifications, because the convolutional neural network (CNN/ConvNet) is the most commonly applied to analyse visual imagery in deep neural networks.

In this model evaluation process, we started from a simple and baseline model, and tuned it in 8 iterations to get the final output model including adding more layers, choosing the right activation function, using the suitable initialization, regularizing the weights, adding dropout between layers, adding batch normalization, using residual layers, using learning scheduler, using image augmentation etc.

In addition, we chose **ReLU** as the activation functions in every convolutional layer and the dense layers except the last dense layer. The last dense layer uses **Softmax** function as the activation function. The **categorical_crossentropy** is applied in the loss function to minimize the errors. The

number of training epochs is only set to 100 because of the limitation of our computational resources.

The final accuracy of our CNN model is 99.40% which is a fantastic number.

2.3.2. Baseline Model

You can find the details of this model in **../source/cnn_train_model_v0.ipynb**

As shown in Figure 10, we used two convolutional layers in this neural network to extract the features from the labelled image dataset and used 2 dense layers to make the classification on the two classes, wear masks and not wear masks.

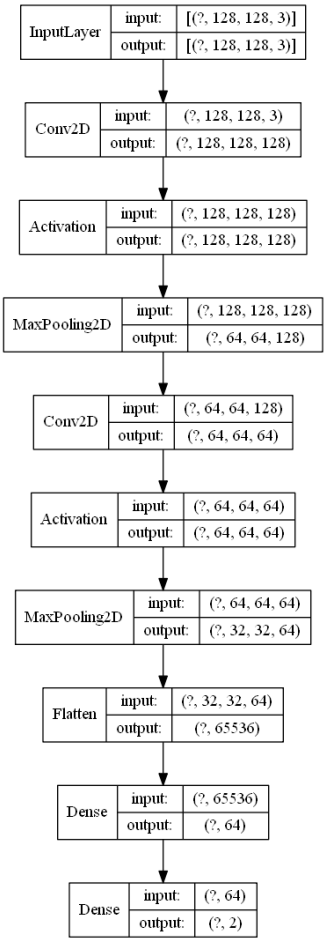

Figure 10 CNN baseline model - Model Plot

As shown in Figure 11, the accuracy of our baseline model on the testing dataset is 93.98% which is already acceptable for a classification model. However, if we look at the loss value diagram, the loss value of the validation rises rapidly and the loss value of training is near zero which means our model is kind of overfitting.

```
Best accuracy (on testing dataset): 93.98%
              precision    recall  f1-score   support

   with_mask     0.9567    0.9195    0.9377       745
without_mask     0.9245    0.9595    0.9417       766

    accuracy                         0.9398      1511
   macro avg     0.9406    0.9395    0.9397      1511
weighted avg     0.9404    0.9398    0.9397      1511
```

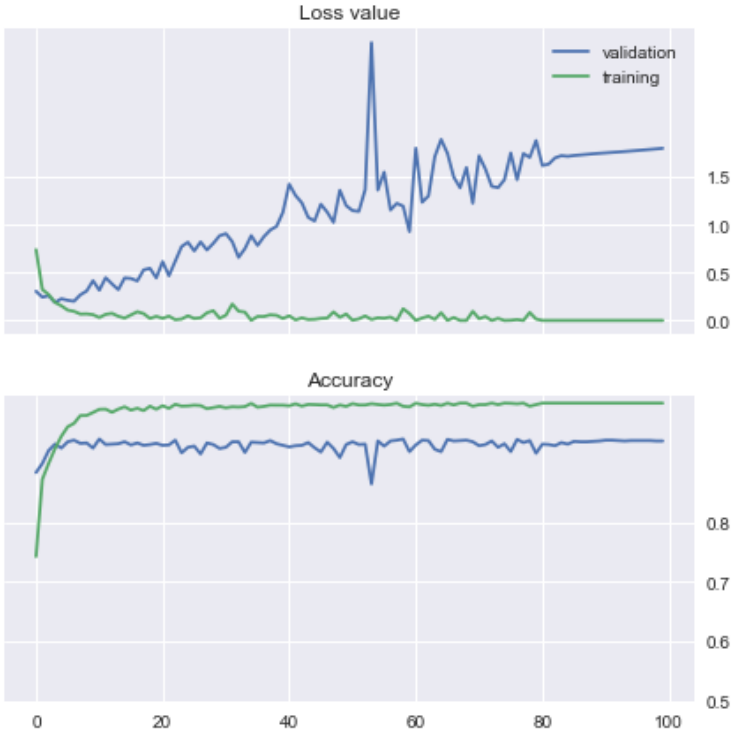

Figure 11 CNN baseline model - Loss and Accuracy

2.3.3. First Iteration - Add More Convolutional Layers

You can find the details of this model in **../source/cnn_train_model_v1.ipynb**

As shown in Figure 12, we added one more convolutional layer in this neural network to extract the features from the labelled image dataset which helps the model to extract more detailed information from the dataset to improve the accuracy.

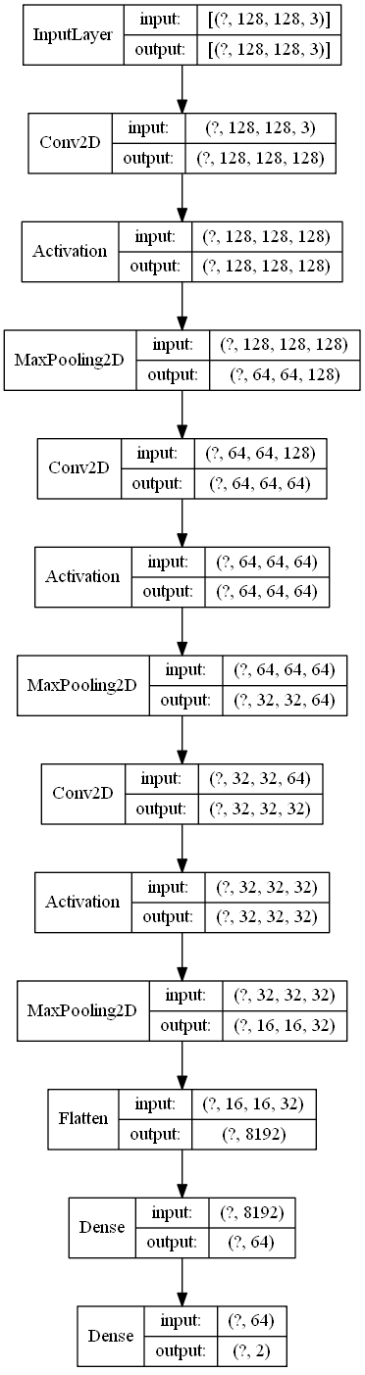

Figure 12 CNN 1st iteration model - Model Plot

As shown in Figure 13, the accuracy of our 1st iteration model on the testing dataset is 95.76% which is higher than the baseline model. But in the loss value diagram, the loss value of the validation still rises rapidly and has no improvement which means our model still faces the issue of overfitting.

```
Best accuracy (on testing dataset): 95.76%
              precision    recall  f1-score   support

   with_mask     0.9658    0.9477    0.9566       745
without_mask     0.9500    0.9674    0.9586       766

    accuracy                         0.9576      1511
   macro avg     0.9579    0.9575    0.9576      1511
weighted avg     0.9578    0.9576    0.9576      1511
```

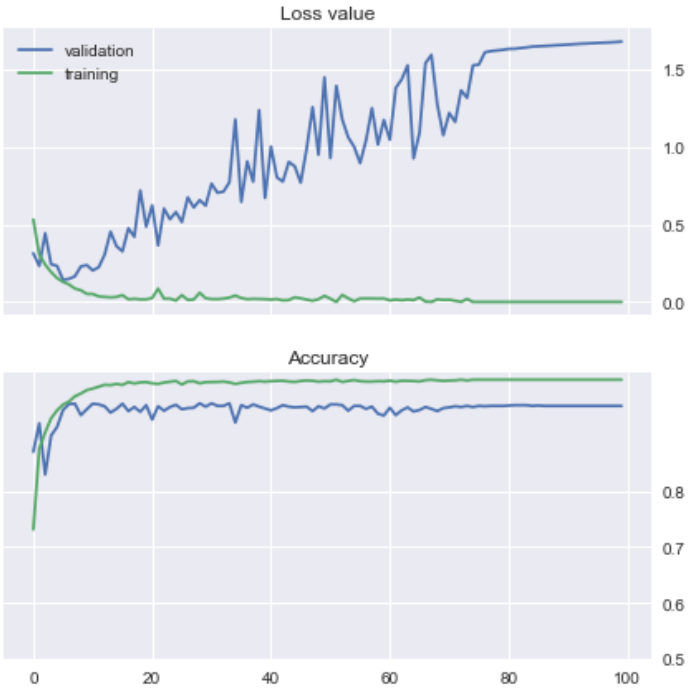

Figure 13 CNN 1st iteration model - Loss and Accuracy

2.3.4. Second Iteration - Use He Weights Initialization

You can find the details of this model in **../source/cnn_train_model_v2.ipynb**

As shown in Figure 14, we used "He weights initialization" as the kernel initializers in every convolutional layer and dense layer. Furthermore, one more dense layer is also added into this neural network to prevent features lost.

```
def createModel():

    ipt = Input(shape=(imgrows, imgclms, channel))
    x = Conv2D(128,(3,3),padding='same', kernel_initializer=he_normal(33))(ipt)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size=(2,2))(x)
    x = Conv2D(64,(3,3),padding='same', kernel_initializer=he_normal(33))(x)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size=(2,2))(x)
    x = Conv2D(32,(3,3),padding='same', kernel_initializer=he_normal(33))(x)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size=(2,2))(x)
    x = Flatten()(x)
    x = Dense(64, activation='relu', kernel_initializer=he_normal(33))(x)
    x = Dense(32, activation='relu', kernel_initializer=he_normal(33))(x)
    x = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=ipt, outputs=x)
```

Figure 14 CNN 2nd iteration model - CreateModel method

As shown in Figure 15, the accuracy of our 2nd iteration model on the testing dataset is still 95.76% which is the same as the first iteration model. But in the loss value diagram, the loss value of the validation is lower than the first iteration model. Although the loss value still rises, using He weights initialization is workable for reducing loss.

```
Best accuracy (on testing dataset): 95.76%
              precision    recall  f1-score   support

   with_mask     0.9608    0.9530    0.9569       745
without_mask     0.9547    0.9621    0.9584       766

    accuracy                         0.9576      1511
   macro avg     0.9577    0.9576    0.9576      1511
weighted avg     0.9577    0.9576    0.9576      1511
```

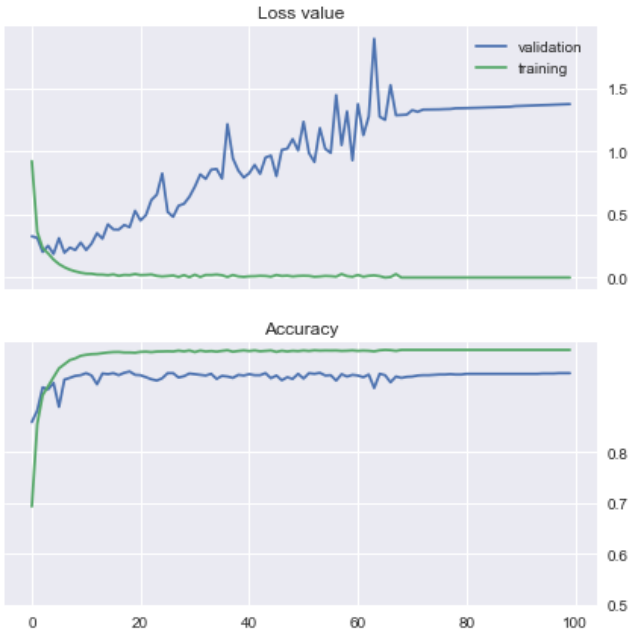

Figure 15 CNN 2nd iteration model - Loss and Accuracy

2.3.5. Third Iteration - Regularize the Weights

You can find the details of this model in **../source/cnn_train_model_v3.ipynb**

As shown in Figure 16, we added “L2 regularization” which is more often used compared with L1 as the kernel regularizer in every convolutional layer and dense layer. A penalty parameter α=0.0001 was chosen in this method.

```
def createModel():

    ipt = Input(shape=(imgrows, imgclms, channel))
    x = Conv2D(128,(3,3),padding='same', kernel_initializer=he_normal(33), kernel_regularizer=regularizers.l2(1e-4))(ipt)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size=(2,2))(x)
    x = Conv2D(64,(3,3),padding='same', kernel_initializer=he_normal(33), kernel_regularizer=regularizers.l2(1e-4))(x)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size=(2,2))(x)
    x = Conv2D(32,(3,3),padding='same', kernel_initializer=he_normal(33), kernel_regularizer=regularizers.l2(1e-4))(x)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size=(2,2))(x)
    x = Flatten()(x)
    x = Dense(64, activation='relu', kernel_initializer=he_normal(33), kernel_regularizer=regularizers.l2(1e-4))(x)
    x = Dense(32, activation='relu', kernel_initializer=he_normal(33), kernel_regularizer=regularizers.l2(1e-4))(x)
    x = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=ipt, outputs=x)
```

Figure 16 CNN 3rd iteration model - CreateModel method

As shown in Figure 17, the accuracy of our 3rd iteration model on the testing dataset is still 96.03% which is a little bit higher than the 2nd iteration model. Furthermore, the loss value of the validation

does not rise and is much lower than the 2nd iteration model. The overfitting issue of our model has been solved mostly by the first 3 iterations but we still can improve the accuracy in the following iterations.

```
Best accuracy (on testing dataset): 96.03%
              precision    recall  f1-score   support

   with_mask     0.9647    0.9544    0.9595       745
without_mask     0.9561    0.9661    0.9610       766

    accuracy                         0.9603      1511
   macro avg     0.9604    0.9602    0.9603      1511
weighted avg     0.9603    0.9603    0.9603      1511
```

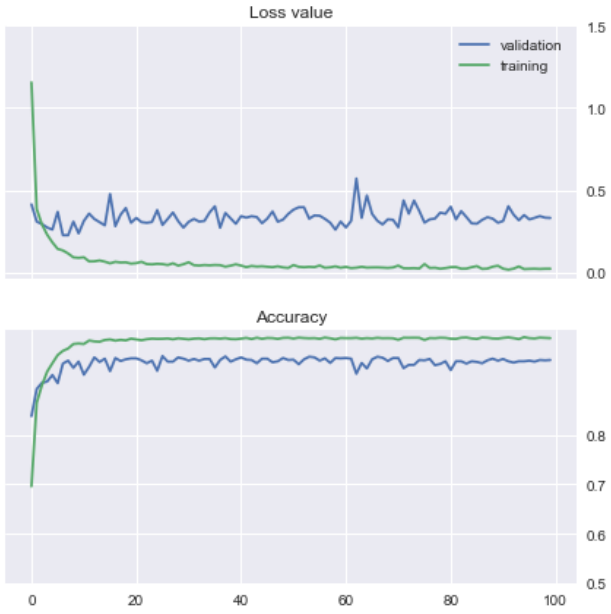

Figure 17 CNN 3rd iteration model - Loss and Accuracy

2.3.6. Fourth Iteration - Add Dropout between Layers

You can find the details of this model in **../source/cnn_train_model_v4.ipynb**

As shown in Figure 18, we added dropout layers with rate=0.5 to randomly drop out neurons from the dense layers.

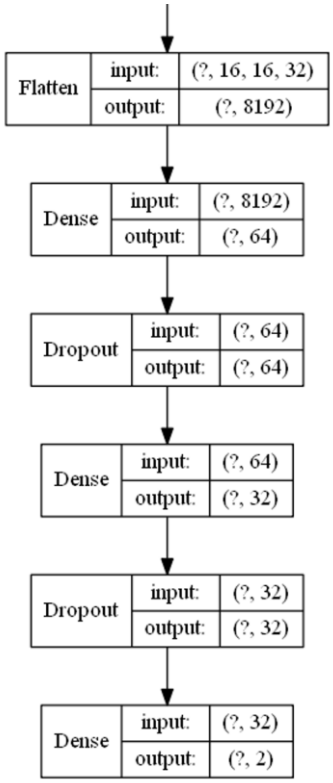

Figure 18 CNN 4th iteration model - Model Plot

As shown in Figure 19, the accuracy of our 4th iteration model on the testing dataset is 96.36% which is a little bit higher than the 3rd iteration model. To a certain extent, adding dropout helps increase the accuracy.

```
Best accuracy (on testing dataset): 96.36%
              precision    recall  f1-score   support

   with_mask     0.9650    0.9611    0.9630       745
without_mask     0.9623    0.9661    0.9642       766

    accuracy                         0.9636      1511
   macro avg     0.9636    0.9636    0.9636      1511
weighted avg     0.9636    0.9636    0.9636      1511
```

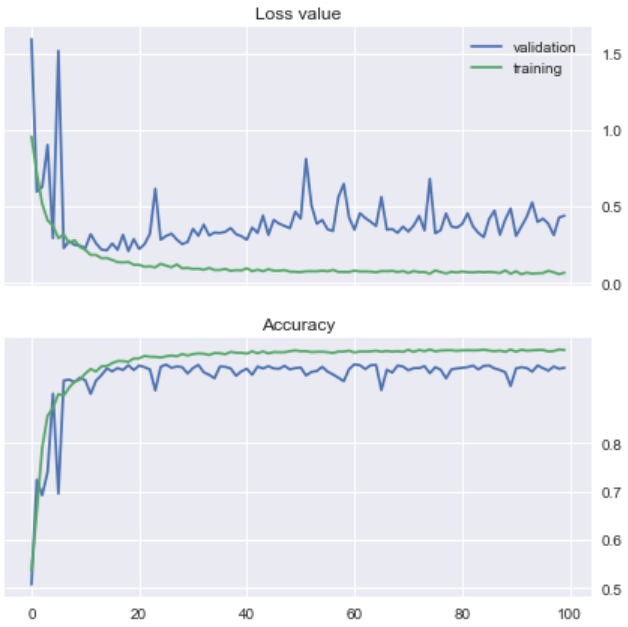

Figure 19 CNN 4th iteration model - Loss and Accuracy

2.3.7. Fifth Iteration - Add Batch Normalization

You can find the details of this model in **../source/cnn_train_model_v5.ipynb**

As shown in Figure 20, we added batch normalization layers after every convolutional layer.

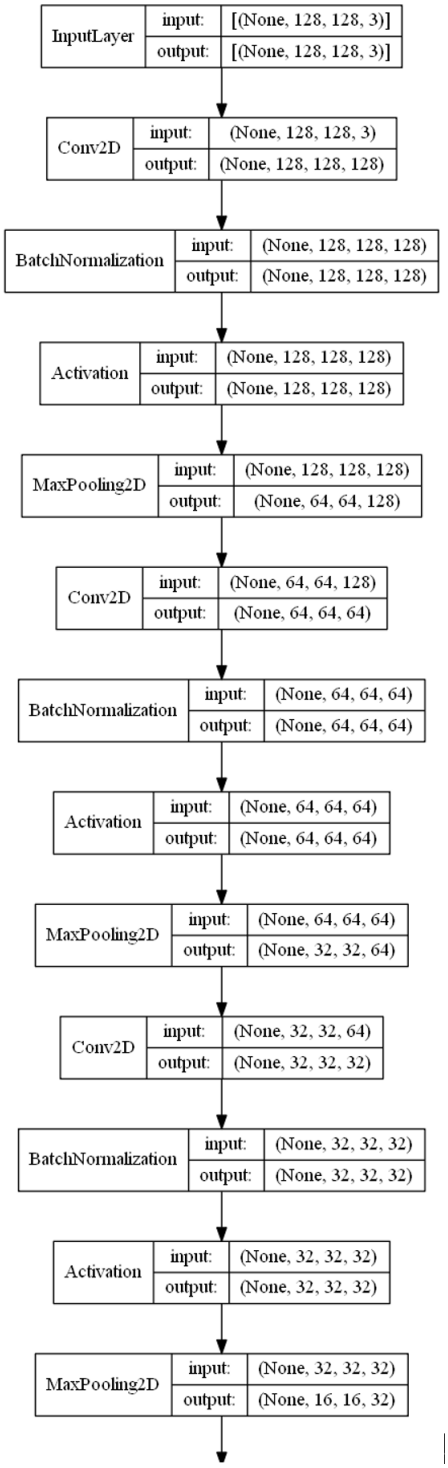

Figure 20 CNN 5th iteration model - Model Plot

As shown in Figure 21, the accuracy of our 5th iteration model on the testing dataset is 97.22% which is higher than the 4th iteration model. In addition, the loss value also falls compared to the 4th iteration model.

```
Best accuracy (on testing dataset): 97.22%
              precision    recall  f1-score   support

   with_mask     0.9795    0.9638    0.9716       745
without_mask     0.9653    0.9804    0.9728       766

    accuracy                         0.9722      1511
   macro avg     0.9724    0.9721    0.9722      1511
weighted avg     0.9723    0.9722    0.9722      1511
```

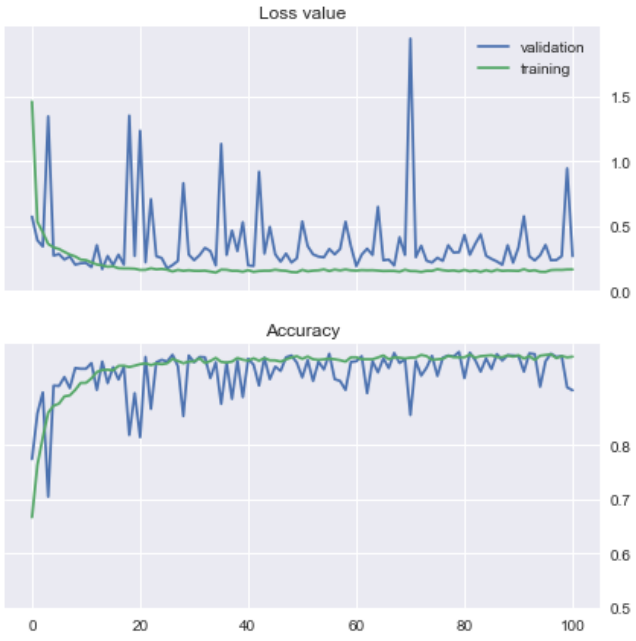

Figure 21 CNN 5th iteration model - Loss and Accuracy

2.3.8. Sixth Iteration - Add Residual Layers

You can find the details of this model in **../source/cnn_train_model_v6.ipynb**

In deep neural networks, the depth of the network would affect the accuracy of a model extremely. As shown in Figure 22, we added much more layers than the 5th iteration models and combined these residual layers as Simp ResBlk and Down ResBlk to represent. In total, we added 3 Simp

ResBlks firstly, 1 Down ResBlks followed by 2 Simp ResBlks secondly, 1 Down ResBlks followed by 2 Simp ResBlks thirdly and one AveragingPooling layer lastly.

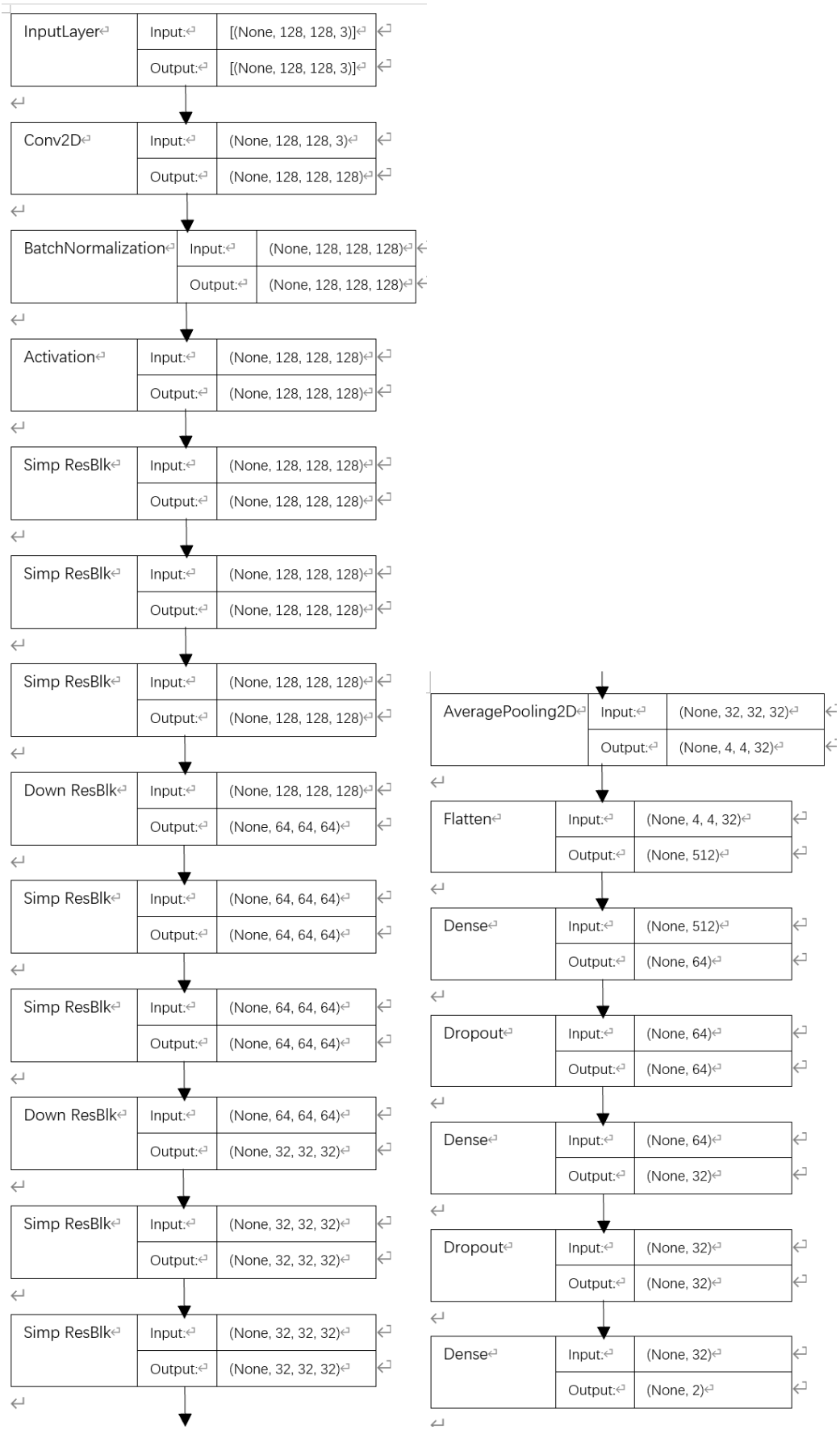

Figure 22 CNN 6th iteration model - Model Plot

As shown in Figure 23, the accuracy of our 6th iteration model on the testing dataset is 98.01% which is higher than the 5th iteration model. In addition, the loss value also falls compared to the 5th iteration model.

```
Best accuracy (on testing dataset): 98.01%
              precision    recall  f1-score   support

   with_mask     0.9812    0.9785    0.9798       745
without_mask     0.9792    0.9817    0.9804       766

    accuracy                         0.9801      1511
   macro avg     0.9802    0.9801    0.9801      1511
weighted avg     0.9801    0.9801    0.9801      1511
```

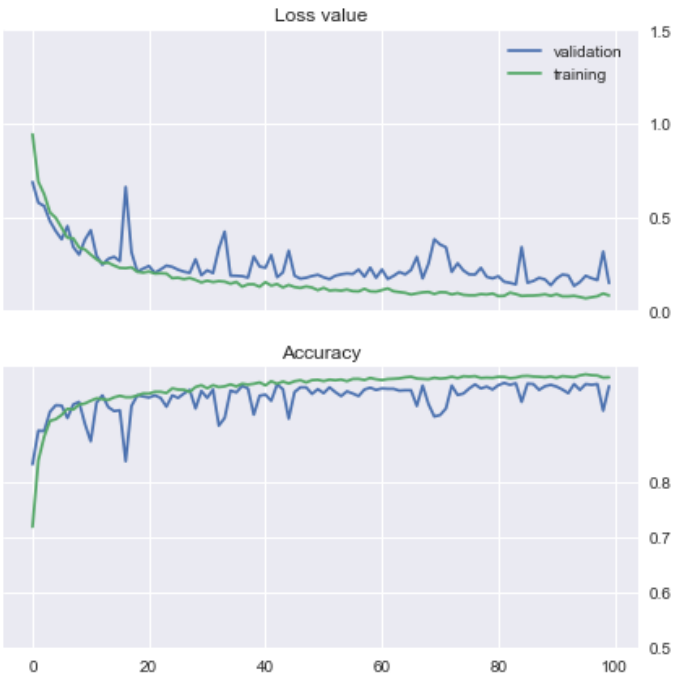

Figure 23 CNN 6th iteration model - Loss and Accuracy

2.3.9. Seventh Iteration - Create Learning Scheduler

You can find the details of this model in **../source/cnn_train_model_v7.ipynb**

In deep neural networks, large learning rates cause the model to converge too fast to a suboptimal solution. As shown in Figure 24, we created our own learning scheduler to move the learning steps fast in the first 50 epochs, but after the 50 epochs, the learning steps become slower and slower, especially the last 10 epochs. The reason we set the boundary like this is because the curves of loss and accuracy from the 6th iteration model are undulate.

```
from tensorflow.keras.callbacks import LearningRateScheduler

def lrSchedule(epoch):
    lr  = 1e-3

    if epoch > 100:
        lr  *= 0.5e-3
    elif epoch > 90:
        lr  *= 1e-3
    elif epoch > 70:
        lr  *= 1e-2
    elif epoch > 50:
        lr  *= 1e-1

    print('Learning rate: ', lr)
    return lr

LRScheduler     = LearningRateScheduler(lrSchedule)
```

Figure 24 CNN 7th iteration model - lrSchedule method

As shown in Figure 25, the accuracy of our 7th iteration model on the testing dataset is 98.41% which is higher than the 6th iteration model.

```
Best accuracy (on testing dataset): 98.41%
              precision    recall  f1-score   support

   with_mask     0.9775    0.9906    0.9840       745
without_mask     0.9907    0.9778    0.9842       766

    accuracy                         0.9841      1511
   macro avg     0.9841    0.9842    0.9841      1511
weighted avg     0.9842    0.9841    0.9841      1511
```

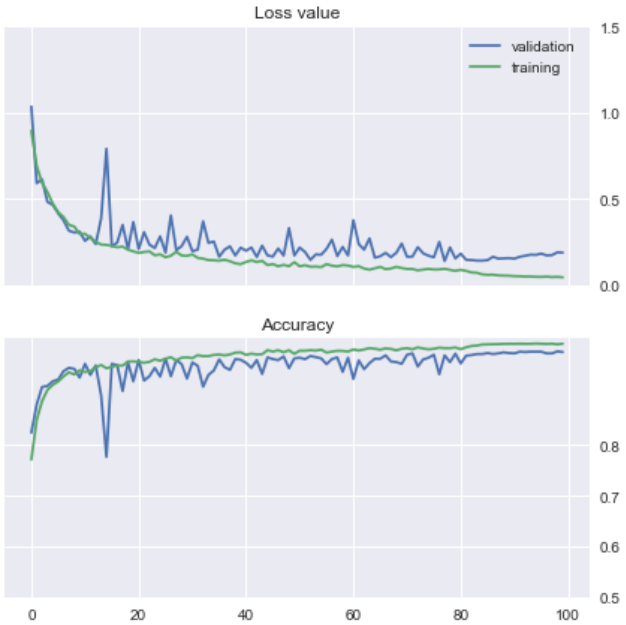

Figure 25 CNN 7th iteration model - Loss and Accuracy

2.3.10. Eighth Iteration - Add Image Augmentation and Change Batch Size

You can find the details of this model in **../source/cnn_train_model_v8.ipynb**

In deep neural networks, the net sees the same set of images every epoch. As shown in Figure 26, we used image augmentation to generate randomly varied images to force the net to learn the features that really matter for classification. In addition, we adjusted the batch size to balance the speed of training and its accuracy.

```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(width_shift_range=0.1,
                             height_shift_range=0.1,
                             rotation_range=20,
                             horizontal_flip=True,
                             vertical_flip=False)


model.fit(datagen.flow(trDat, trLbl, batch_size=16),                    # Training data, Training label
          validation_data=(tsDat, tsLbl),   # Validation data and label
          epochs=100,                       # The amount of epochs to be trained
          verbose=1,
          steps_per_epoch=len(trDat)/16,
          shuffle=True,                     # To shuffle the training data
          callbacks=callbacks_list)         # Callbacks to execute the checkpoints
```

Figure 26 CNN 8th iteration model - Image Augmentation and Batch Size Adjustment

As shown in Figure 27, the accuracy of our 8th iteration model (the final model) on the testing dataset is 99.40% which is highest among all the iteration models.

```
Best accuracy (on testing dataset): 99.40%
              precision    recall  f1-score   support

   with_mask     0.9907    0.9973    0.9940       745
without_mask     0.9974    0.9909    0.9941       766

    accuracy                         0.9940      1511
   macro avg     0.9940    0.9941    0.9940      1511
weighted avg     0.9941    0.9940    0.9940      1511
```

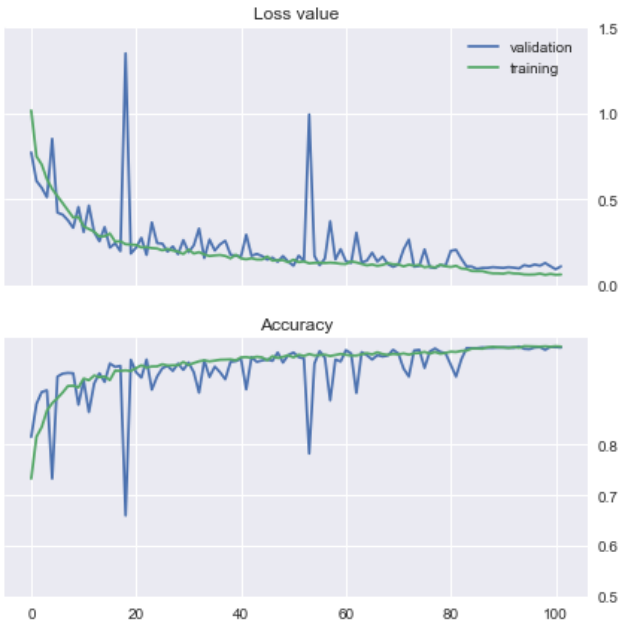

Figure 27 CNN 8th iteration model - Loss and Accuracy

2.4. Comparison of Models

Below diagram shows the accuracy of each model. The worst-behaved model among these three is KNN and it is also very hard to tune hyperparameters to improve its accuracy. Despite the low accuracy of SVM in the first few iterations, the final output of it is still acceptable as a machine learning model. Compared to KNN and SVM models, the accuracy of CNN is significantly higher. In particular, the accuracy of the CNN final iteration reaches over 99%.

Therefore, we chose CNN as the final model in our system.

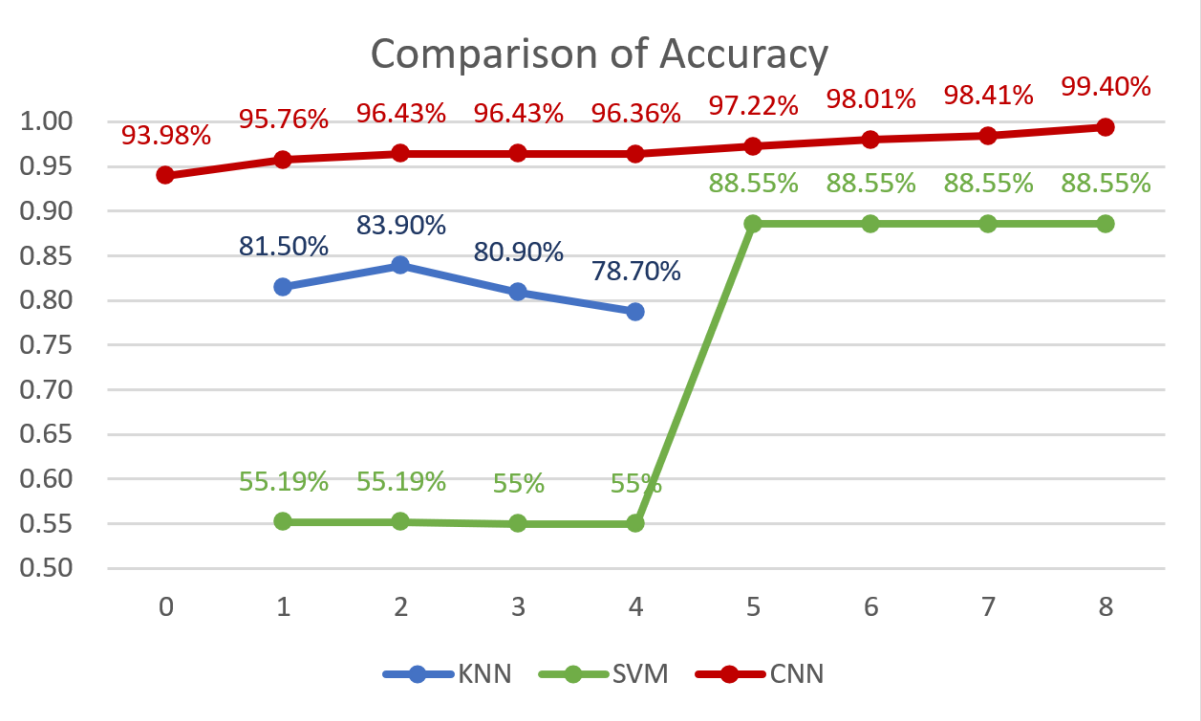

Figure 28 Comparison of Accuracy Diagram

3. System Development & Implementation

3.1. System Architecture

3.1.1. Techniques

1) Flask: webhook service
2) Telegram bot: message alert sender
3) OpenCV: computer vision library
4) Keras: Python artificial neural networks library
5) Sklearn: Python machine learning library
6) Numpy: to process image data

3.1.2. Overview

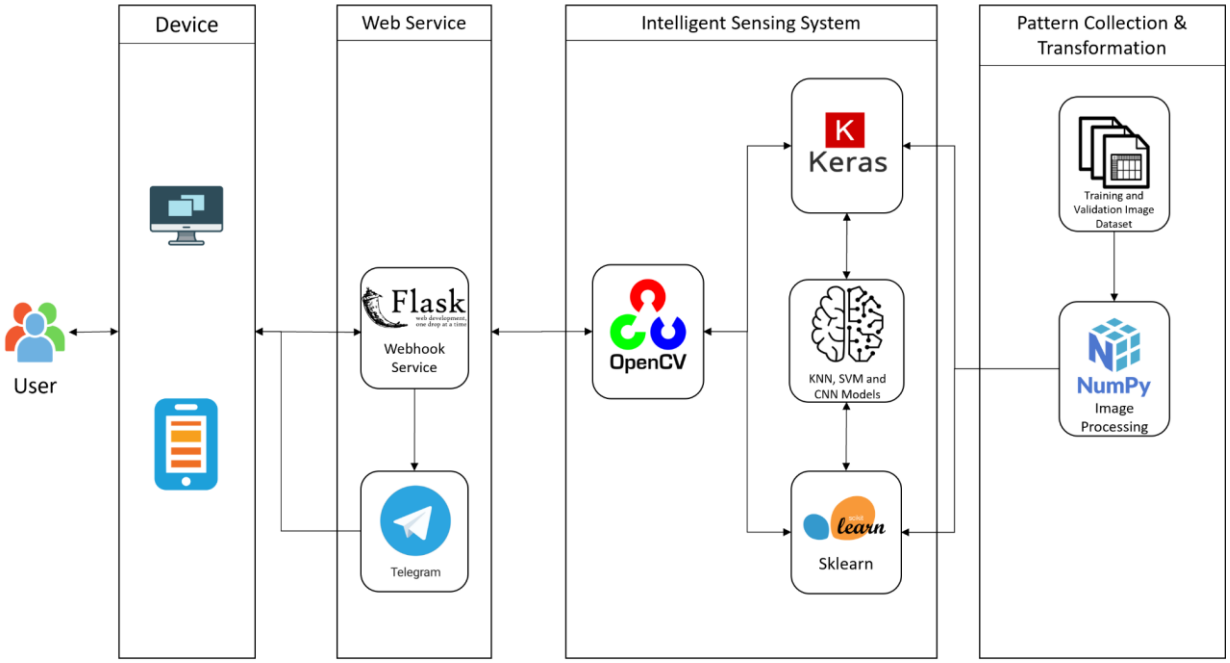

Figure 29 DetectiveMask System - System Architecture

In order to build up a system that consists of front-end and back-end and can integrate with machine learning and artificial neural networks libraries, as a result, Flask, a python-based web framework, is decided to use this project.

Our eco-system is constituted by four parts. Firstly, we used NumPy to pre-process the training and validation image dataset. It provides float32 data to the models instead of image files. Secondly, we leveraged scikit-learn which is a python library for machine learning to build KNN and SVM models or Keras which is another python library for artificial neural networks library to build CNN model to classify or identify if a user is wearing a mask based on the image. Thirdly, users use their camera to

take a picture or video on the front end web page developed by Flask to transfer the picture or video to the backend. Lastly, in the backend, we used the OpenCV vision library to identify the human faces and feed them into the models that we built before. The classification results would be posted to the frontend and we used Telegram bot to send the alert messages if the system detects someone is not wearing a mask.

3.2. Data Sources

3.2.1. Labelled RGB images

We will be using a labelled datasource which is available on Kaggle. This data source consists of 7553 RGB images in 2 folders. These images are labeled with_mask and without_mask.

Figure 30 With mask data set

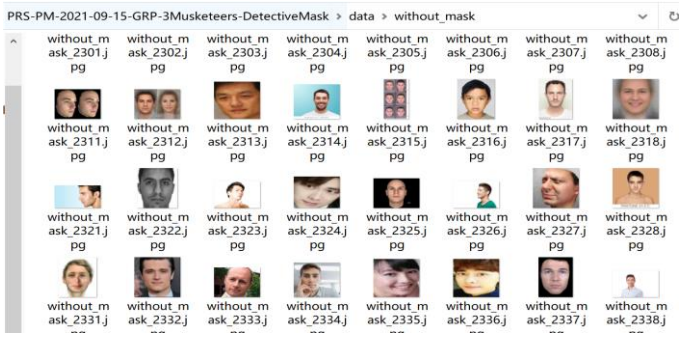

Figure 31 Without mask data set

The data source can be found here: https://www.kaggle.com/omkargurav/face-mask-dataset.

3.3. Sensing System

3.3.1. Overview

The sensors we used in this system are the cameras on the users' computers and mobile phones. The benefit of utilizing cameras is that it is the most common sensor around us, and users do not need to buy additional devices.

The pattern in our system is images. Before feeding images into our models, pre-processing the images is the necessary step. As shown in Figure 32 and 33, we transformed the image files into float32 format before training and predicting.

```
trDat           = trDat.astype('float32')/255
tsDat           = tsDat.astype('float32')/255
```

Figure 32 Image Data Transformation - Before Training

```
faces_preprocessed = np.array(faces_dict["faces_list"])
faces_preprocessed = faces_preprocessed.astype('float32')/255
preds = model.predict(faces_preprocessed)
```

Figure 33 Image Data Transformation - Before Predicting

3.3.2. Face Recognition

In the real phase of development, we found out that the system needs human faces instead of full scale images or videos to make classifications. Unlike in the training phase, the training and validation image dataset is well-prepared and we only feeded images into models. Therefore, recognizing human faces and only feeding human faces into the prediction became a problem that we needed to solve and continue our development.

OpenCV, the most popular computer vision package, was chosen to handle face recognition finally. As shown in Figure 34, we used a cascade classifier which is embedded in the OpenCV to identify the faces.

```
import cv2

def load_cascade_detector():
    cascade_path = os.path.dirname(cv2.__file__) +
    "/data/haarcascade_frontalface_alt2.xml"
    face_detector = cv2.CascadeClassifier(cascade_path)
    return face_detector

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
face_detector_model = load_cascade_detector()
faces = face_detector_model.detectMultiScale(gray,
                                             scaleFactor=1.05,
                                             minNeighbors=4,
                                             minSize=(40, 40),
                                             flags=cv2.CASCADE_SCALE_IMAGE,
                                             )
```

Figure 34 Face Recognition - OpenCV

Furthermore, we also used OpenCV to write a bounding box around faces with confidence to highlight our classification results to users. In Figure 35, this is the python code where we realized this function. And in Figure 36, this is how the final output looks like. You can find the photo in ../app/static/images/one_person.jpg

```
def write_bb(mask_or_not, confidence, box, frame):
    (x, y, w, h) = box
    color = (0, 255, 0) if mask_or_not == "Mask" else (0, 0, 255)
    label = f"{mask_or_not}: {confidence}%"

    cv2.putText(frame, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, color
    , 2)
    cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
```

Figure 35 Face Bounding Box - OpenCV

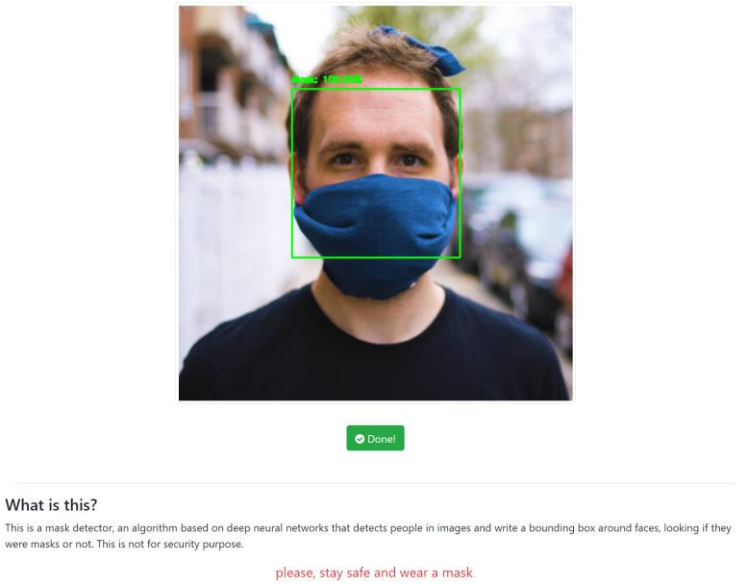

Figure 36 Mask Detection for One Person- UI

Besides the mask detection for one person, we added multi-faces detection functionality into our system as shown in Figure 37. Basically, we used openCV to recognize all the faces in the image or video, feeded them all into a dictionary and make the classifications on all faces from the dictionary.

```
for rect in faces:
    (x, y, w, h) = rect
    face_frame = clone_image[y:y + h, x:x + w]
    # preprocess image
    face_frame_array = preprocess_face_frame(face_frame)

    faces_dict["faces_list"].append(face_frame_array)
    faces_dict["faces_rect"].append(rect)

if faces_dict["faces_list"]:
    faces_preprocessed = np.array(faces_dict["faces_list"])
    faces_preprocessed = faces_preprocessed.astype('float32')/255
    preds = model.predict(faces_preprocessed)
```

Figure 37 Multi-faces Recognition - OpenCV

The final output in the web page looks like as shown below. You can find the photo in ../app/static/images/family.jpg

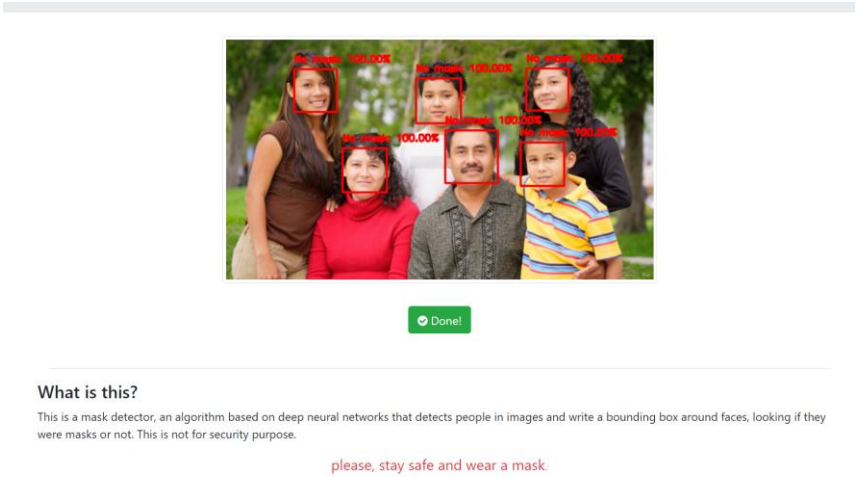

Figure 38 Mask Detection for Many People - UI

3.4. Web Application

3.4.1. Overview

Web application uses Flask framework along with CSS and Javascript for these functions:

1) user interface for users to stream real-time video using web-cam for mask detection.
2) user interface for users to upload an image for mask detection.
3) user interface for users to upload a video for mask detection.

3.4.2. User Interfaces

Figure 39 shows the landing page of the application. It is also the web page where users can feed a live feed, web cam, for mask detection.

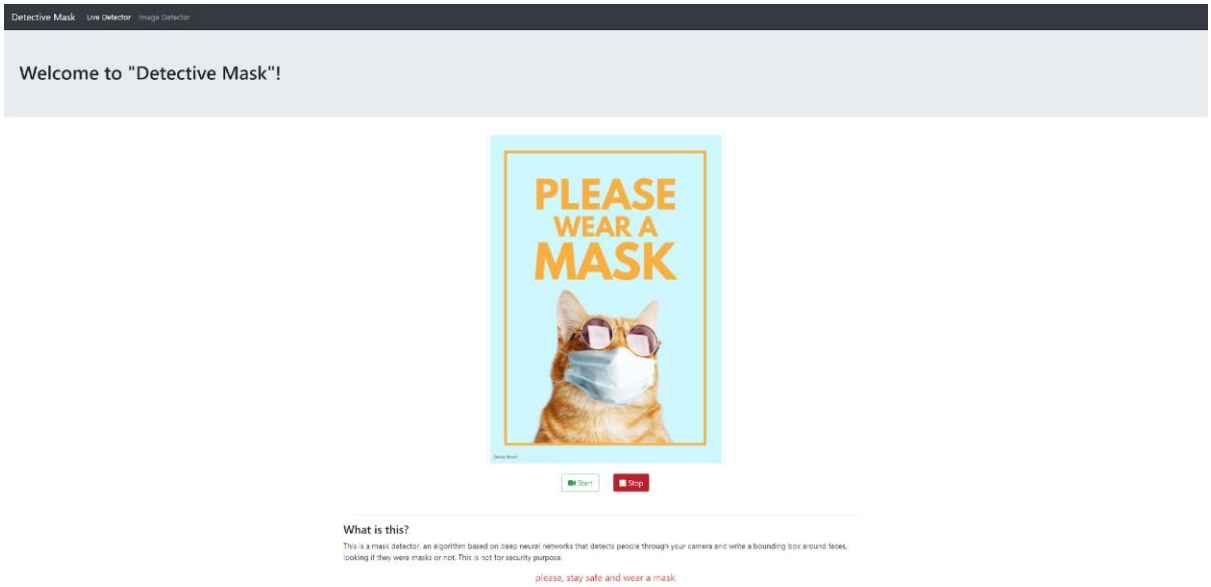

Figure 39 Mask Detection for live feed - UI

Figure 40 shows the user interface for users to upload an image for mask detection. Users can upload an image to test on the model.

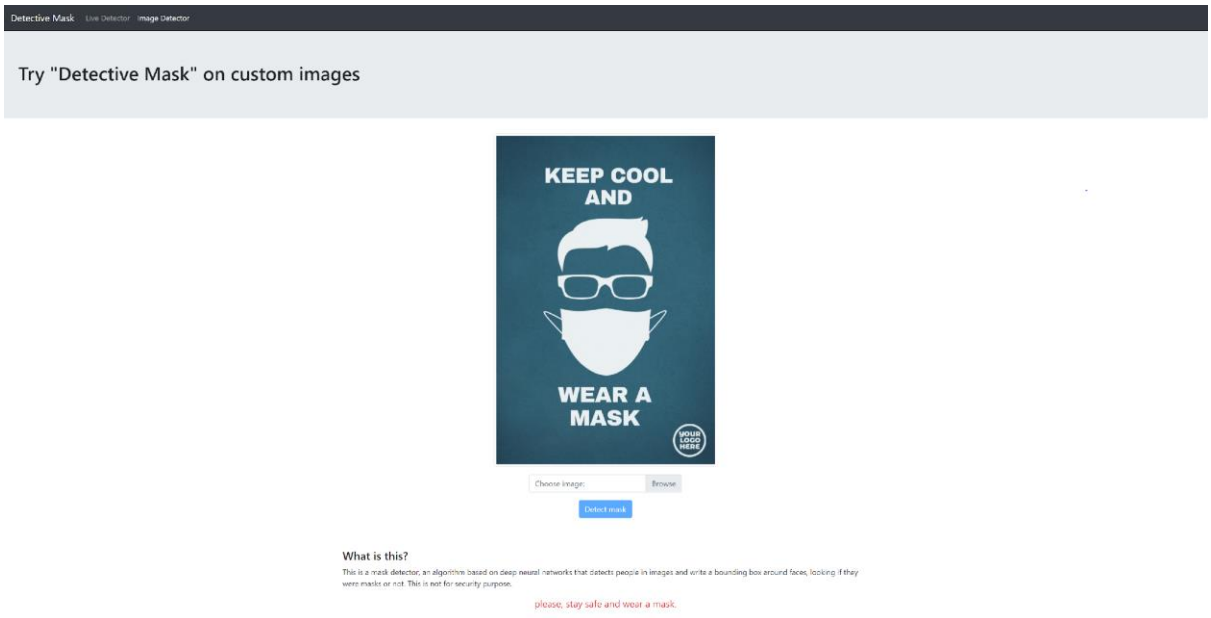

Figure 40 Mask Detection for image - UI

Figure 41 shows the user interface for users to upload an mp4 video for mask detection. Users can upload a small size video to test on the model. You can find the video in ../app/static/uploads/people-wearing-face-mask-empty-street-covid19.mp4

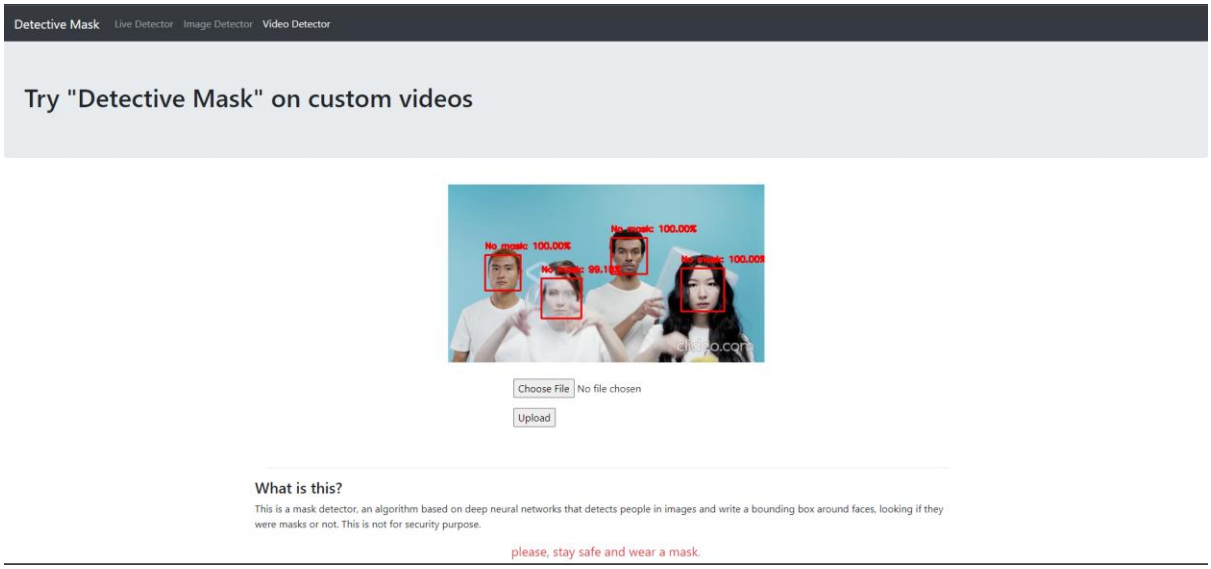

Figure 41 Mask Detection for video - UI

3.4.3. Alert System

For our proof of concept, we used telegram as an alert system. To do so, we first had to create a telegram bot to relay this information to the group. This is because telegram is widely used and can support various types of devices (desktop, phones, tablets). The alert system will provide the user with the image and time it was recorded.

3.4.3.1. Creating telegram bot

A telegram bot can be created via the telegram application itself.

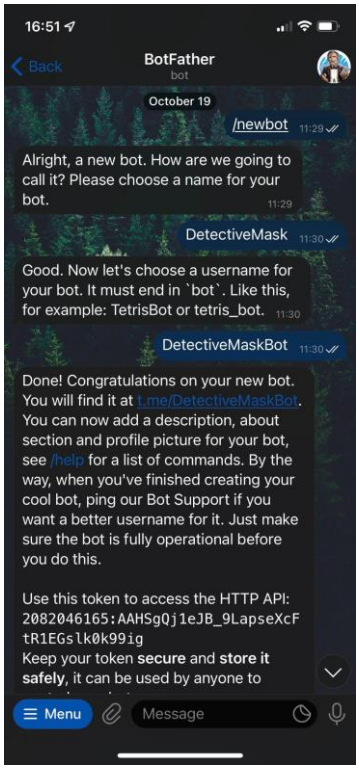

Figure 42 Creating telegram bot

Once the bot is created, telegram will provide us with an API key which we will use to push the alerts.

3.4.3.2. Telegram alert system

When a subject who is not wearing a mask is detected, our system will automatically send a notification to the authorities to take an action. In our proof of concept, this works for both images and video. For video, we implemented a 30 second timeout between frames to prevent notification spamming.

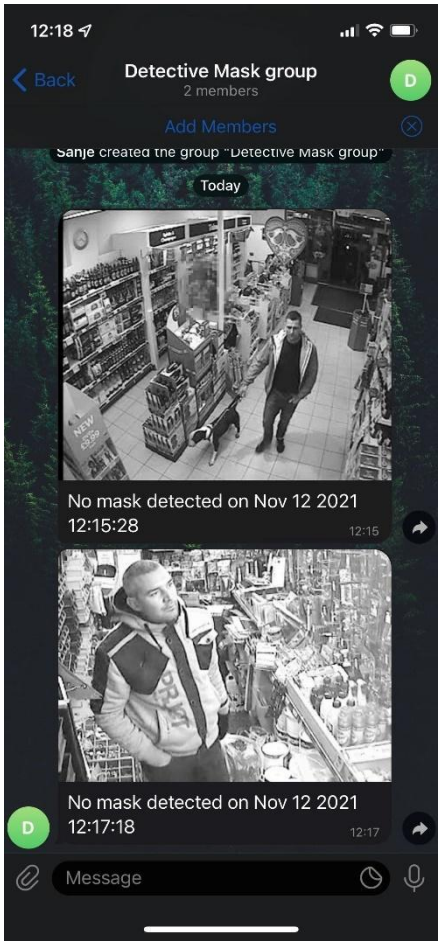

Figure 43 Telegram chat group

4. Findings and Discussions

4.1. Challenges

4.1.1. Balancing work and project

The main challenge for our team is finding time to have meet-ups for project discussion because we have different work schedules (e.g. on different client sites, doing overtime etc) .

4.1.2. First time using Flask framework

This is the first time we are using the Flask framework thus it took us some time to figure out how to use it and create a base framework for the application.

4.1.3. Time taken to train the model

In order to improve the model, a few iterations were carried out to find the most suitable model for our application. However, due to lack of the optimal hardware requirements to train the model, it took hours for one iteration to be complete.

4.2. Future Improvements

If we have a longer timeframe to work on this project, we would have work on these areas:

4.2.1. Alert system

Other forms of alerts such as SMS can be implemented depending on the situation. For implementation in big shopping centres or open public spaces, we can implement zoning in our alerts so that the authorities will be able to go straight to the right area. We can also implement a database to help keep track of repeated offenders. For example, someone may have been caught and approached by staff. Once the staff has left, they proceed to remove their mask again. These people should be tracked as repeat offenders.

4.2.2. Offline/Online learning of the model

A pipeline could be built to collect, analyze, train, test and validate the dataset at regular intervals (offline) or at an incremental manner by continuously feeding the data (online) to allow the model to learn about new data. Thus, new data could be predicted appropriately as well.

4.2.3. Multi-faces recognition

Currently, the multi-faces recognition built by OpenCV is unstable as such there are times where it could not detect some faces in the same image or frame. We can write our own face recognition algorithm to replace the OpenCV one after we take the Graduate Certificate in Intelligent Sensing Systems.

4.2.4. Video format

As of now, only mp4 video upload is supported. We can add more supported video formats if there is a longer timeframe to work on this project.

4.3. Conclusion

Our team has learned quite a few things while working on this project.

Technically, we picked-up Flask and OpenCV, and combined a few technical skills like Keras and Sklearn we learnt in the course to build a complete intelligent pattern recognition system. We built three models including KNN, SVM and CNN and tuned them in several iterations to come out with the best accuracy. In this process of developing models, we had a more sophisticated understanding on how the pattern recognition works, how to apply the models in real world problems and how to fix the overfitting issues in practice. After comparing these three models, the extremely good accuracy of CNN shows us why it is dominating the image analysis area. That is because it automatically detects the important features without any human supervision.

In addition, while completing this project, it shows us the importance of pre-processing the training dataset, choosing the suitable model that depends on the problem we need to solve and not building a powerful net in one go.

Overall, it was a truly enjoyable process. We get to learn from one another and apply what we learned onto a solution which resolves an everyday problem in the current Covid-19 situation.

APPENDIX OF REPORT A

-

Project Proposal

GRADUATE CERTIFICATE: Pattern Recognition Systems (PRS)

PRACTICE MODULE: Project Proposal

Date of proposal: 7 Sept 2021

Project Title: Detective Mask

Group ID (As Enrolled in LumiNUS Class Groups):

3Musketeers

Group Members (Name, Student ID, Email):

Anita Koo Shi Qi, XXXXX480B, anita.koo@ncs.com.sg

Hong Xiaohui, S9476943D, imhongxiaohui@gmail.com

Sanjeven Ramakrishnan, XXXXX938E, sanjeven.ramakishnan@ncs.com.sg

Sponsor/Client: *(Company Name, Address and Contact Name, Email, if any)*

NUS ISS,

25 Heng Mui Keng Terrace Singapore 119615

Background/Aims/Objectives:

During the Covid-19 pandemic, it is mandatory for persons who are 6 years old and above to wear a mask when leaving their homes. However, there are people still not wearing a mask when they are outside. And it is very difficult for staff/authorities to supervise the public to keep wearing a mask especially in a large-scale event. The project's objective is to create an application to detect people not wearing a mask via video stream and images and thus notifying the staff/authorities who are near-by.

Project Descriptions:

1. **Solution:**

    - A web application based on machine learning algorithms and deep neural networks would be developed.
    - Real-time video streaming mode would be provided which would allow users to detect people who are masked or not by camera in real-time.
    - Image mode would be provided which would allow users to upload images to detect people inside with masks or not.
    - Bounding boxes around faces with the accuracy of people wearing masks would be shown in video streaming and images.
    - Dataset contains a few thousands images of people wearing and not wearing masks.
    - Several trained models with the progressive approaches that improve the final accuracies.

    Good to have Feature

    - A notification function that messages to users with the screenshot of people who don't wear masks.

2. **Why is our solution good?**

    - Lesser manpower required to check if people are adhering to the rule
    - Minimize the spread of Covid-19 as wearing a mask can prevent transmission of the virus thus identifying those not wearing one, helps to protect others.

3. **Deliverables and Success Criteria:**

    - A runnable pattern recognition system.
    - Dataset as described in solution.

- Final report that describes techniques the 3Musketeers have used, system design, models, system performance and findings and discussions.
- Repository that contains codes and models.
- A video presentation file that contains a 10 -15 mins presentation
- Slides of two presentations.

4. **Schedule Outline:**

- From 15 Sept 2021, the 3Musketeers will start to work on this project after the proposal is approved.
- On 12 Oct 2021, the 3Musketeers needs to make the first presentation in which they outline the goals of their projects along with details of the data resources required/available, techniques/tools used, progress, etc.
- On 14 Nov 2021, the 3Musketeers needs to handover the runnable pattern recognition system to NUS ISS.

5. **Project Strategy:**

- A traditional waterfall approach would be adopted in this project. The 3Musketeers would follow the project timeline to finish the system on time.
- Risks can be assumed are
    - Computation resources are limited that images dataset would take a long time to get trained
    - False positive (e.g. someone wearing a shirt with a face on it)

Team Formation & Registration

Team Name: 3Musketeers
Project Title (repeated): ISS Project – Intelligent Mask Detection and Alert Generation Platform
System Name (if decided): Detective Mask
Team Member 1 Name: Hong Xiaohui
Team Member 1 NRIC Number: SXXXX943D
Team Member 1 Contact (Mobile/Email): +65-97805666 / imhongxiaohui@gmail.com
Team Member 2 Name: Anita Koo Shi Qi
Team Member 2 NRIC Number: SXXXX480B
Team Member 2 Contact (Mobile/Email): +65-96216596 / anita.koo@ncs.com.sg

Team Member 3 Name: Sanjeven Ramakrishnan
Team Member 3 NRIC Number: SXXXX938E
Team Member 3 Contact (Mobile/Email): +65-98153769 / sanjeven.ramakrishnan@ncs.com.sg

APPENDIX OF REPORT B

-

Installation and User Guide

Requirements

Prerequisite

- Computer with internet access
- Python 3.7 installed

Recommended browsers

Our screening system supports the following browsers:

- Microsoft Edge version 90 and above
- Google Chrome version 90 and above
- Opera 75 and above

System Overview

Our mask detection system is targeted at businesses who currently have employed safe distancing officers who make sure patrons wear their masks at all times. In our current situation in Singapore, wearing masks is mandatory and is enforced. Safe distancing officers are employed to patrol and make sure patrons have their masks on at all times. Our system would like to reduce human intervention so that the majority of these safety officers can be re-deployed to work on other sectors which require more manpower. One of such sectors is covid-19 helplines which have a long waiting time due to lack of manpower.

Besides detection, our system will provide the authorities with an alert for them to act on.

Deployment

Our system is deployed to a windows server. In order to run our system locally, you would need python 3.7 installed along with the following packages. These packages can also be installed via the requirements files provided.

To get started, we can download the source code at https://github.com/xiaohuihong/PRS-PM-2021-09-15-GRP-3Musketeers-DetectiveMask.git or via the git clone command below:

```
gh repo clone xiaohuihong/PRS-PM-2021-09-15-GRP-3Musketeers-DetectiveMask
```

Navigate to \app folder. Install the required packages and start up the server locally with the following commands:

```
pip install -r requirements.txt

python __init__.py
```

To setup the alert notification system, join the chat group using the following link:

https://t.me/joinchat/VtSxgK0B-i84ZWE9

or by scanning the QR code below:

User interface

Once the server is running, use your preferred browser to visit the link localhost:8000. We would recommend using Google chrome. We have two main sections to our application, live detector and image detector.

Live detector

This is the page users will first see when they access the application. Here, they will also be able to start using the system. This can be done by clicking the start button to start the live detection.

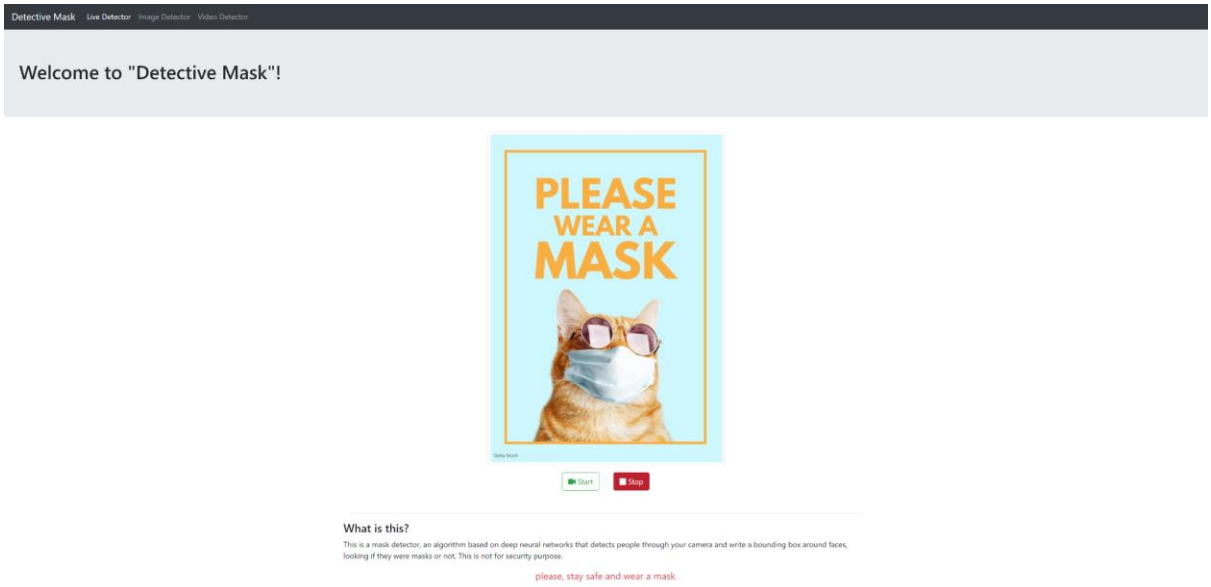

Image 1: Landing page/ live detector

Image detector

In this page, users will be able to manually input a still image to test the model. Users can do this by firstly browsing to the image using the browse button and then to press the detect mask to run the prediction.

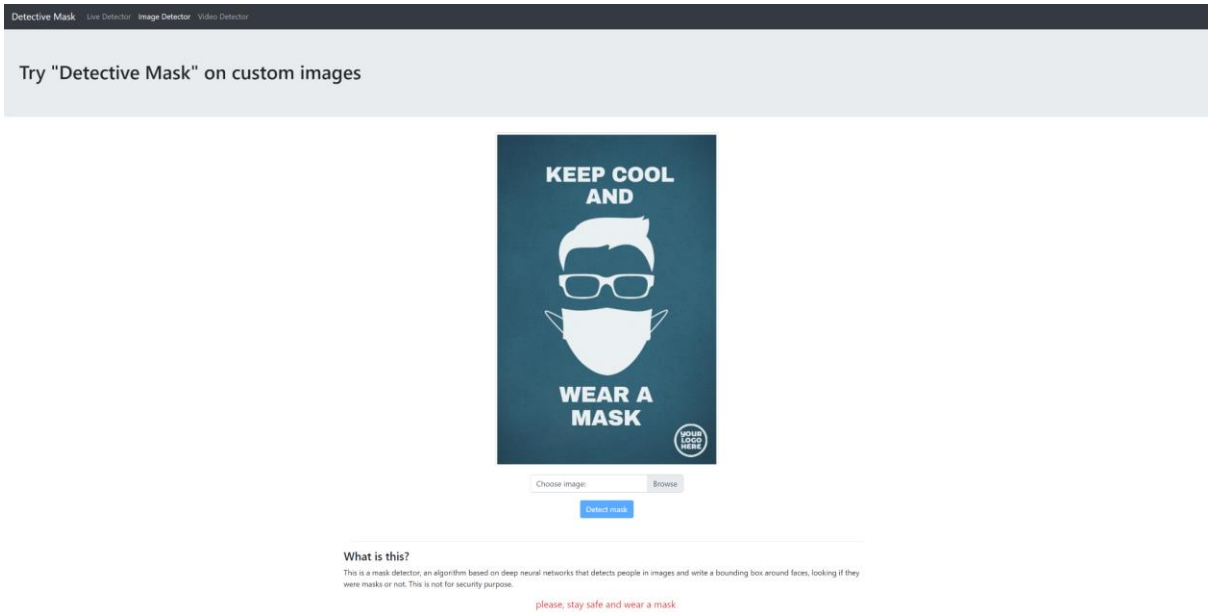

Image 2: Image detector

Video detector

In this page, users will be able to manually input a video recording to test the model or to perform analysis. Users can do this by firstly browsing to the video file using the browse button and then to press the upload button to run the prediction.

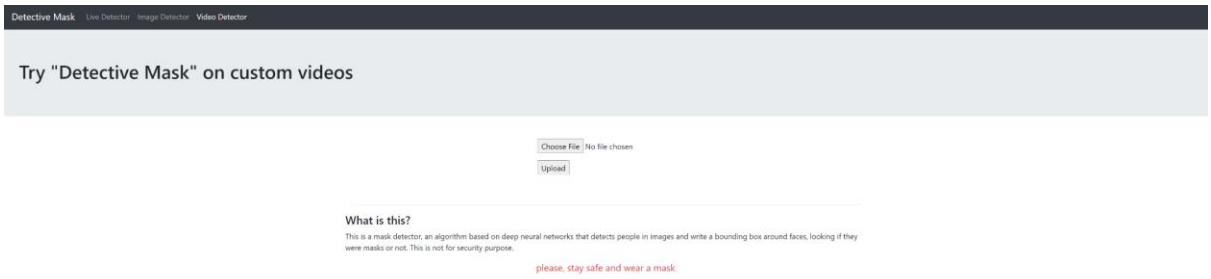

Image 3: Video detector

Alert System

This proof of concept is currently using telegram as the mode of communication. Once a face is detected without a mask for either video or image detection, the notification will be automatically pushed to the telegram group. For video detection, there is a 30 second delay between frames to prevent notification spamming.

Image 4: Chatbot alerting users in the group