Table of Contents

1. Mask Identification Model

1.1 Introduction

It was only 5 years ago, where a deadly virus sparked millions of people to be infected and more than 7 millions deaths reported worldwide (WHO, 2024). This virus was the corona virus also known as COVID-19. It caused many countries to be at a standstill, from its bustling streets filled with tourists, it became like a ghost town where many people were made to stay home and travelling halts (Straits Times, 2020). These social distancing efforts helps to control the pandemic virus. There was also a study reported that having a facial protection such as a mask that covers the nose and mouth does help in reducing the spread of the virus (Goydoy, 2020). Thus, mask was made mandatory during this period.

With mask being made mandatory, there are surveillance systems created that help to detect whether a person is wearing a mask. These systems are able to aid in catching those that are not wearing a mask and would be given a warning. Therefore, the machine learning model that was built got to be accurate in detecting the presence of a face mask, so to give an accurate reading.

1.2 Objective

The objective for this study is to create a model that would be able to take an individual's image and detect whether if there is a presence of a face mask. The model was built using the convolutional neural network (CNN) which is primarily used for image recognition and processing (ARM).

1.3 Data Preprocessing

Dataset that were used for training and testing the model were taken from the WIDER Face dataset and the Masked Faces (MAFA) dataset. The WIDER Face dataset was used as a benchmark for face detection test and it was also used to train the model for the without mask category. The other dataset, MAFA, were images of individuals with mask. This was used to train the model in detecting the presence of a face mask.

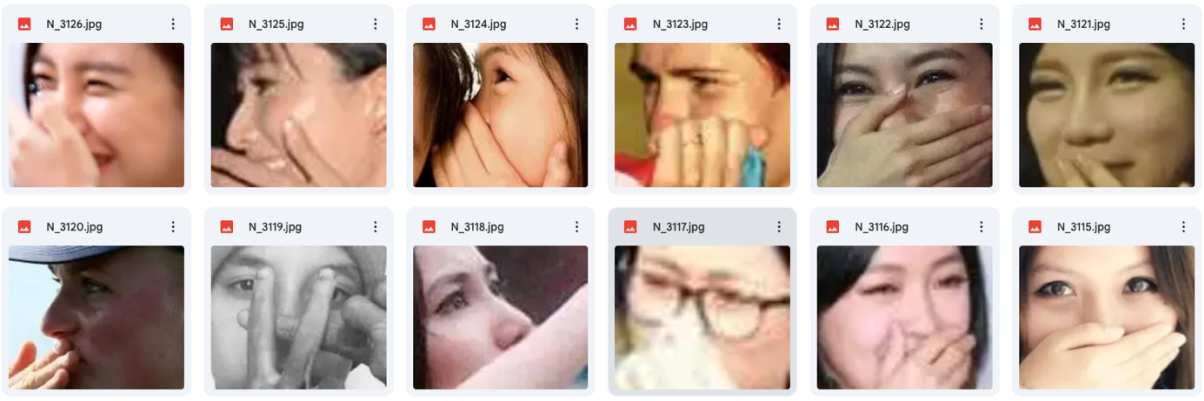

Figure 1: Sample images from WIDER Face dataset

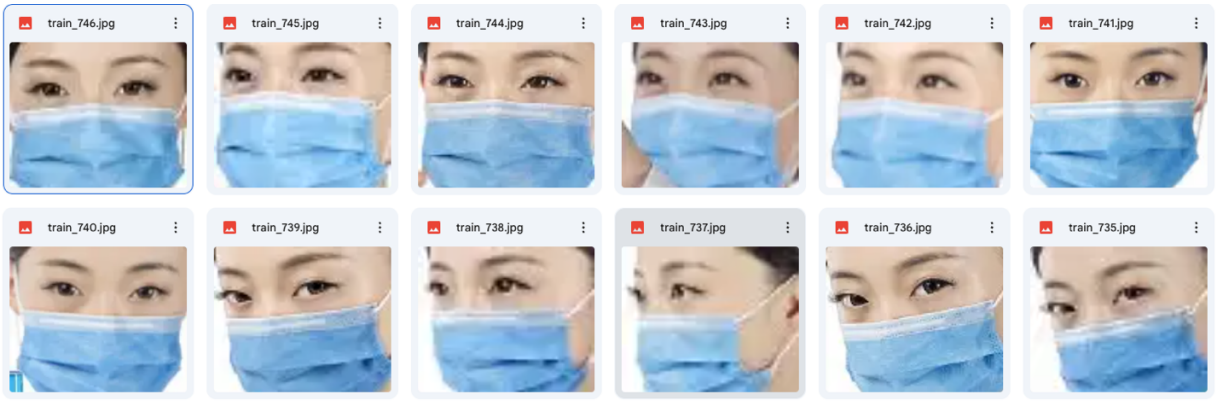

Figure 2: Sample Images from MAFA dataset

By performing the len() function, I was able to get the number of items in each dataset. In the train file, the dataset consist of 154 images of person with mask while person without mask consists of 159 images.

```
print("Number of mask images:", len(files_mask))
print("Number of without mask images:", len(files_no_mask))
```

```
Number of mask images: 154
Number of without mask images: 159
```

Figure 3: Length of training dataset

After finding the length of the dataset, categorisation/labelling was done. Images on person wearing a mask are labelled as 1 while person not wearing a mask are labelled as 0. The labelling was given like a binary so that the system only recognise true and false.

```
mask_label = [1] * 154
no_mask_label = [0] * 159
print(len(mask_label))
print(len(no_mask_label))
```

```
154
159
```

```
# setting labels

labels = mask_label + no_mask_label
print(len(labels))
print(labels[0:5])
print(labels[-5:])
```

```
313
[1, 1, 1, 1, 1]
[0, 0, 0, 0, 0]
```

Figure 4: Categorising the labels

1.4 Dataset Testing

Before preparation of the data and building the CNN model, I first tested whether if the image data in the dataset by using the plot function on the matplotlib library. Both the mask and without mask images can be plotted and produce the results as shown in figure 5 and 6.

```
# mask image
img = mpimg.imread(train_data_path+'/mask/'+'train_00000020.jpg')
imgplot = plt.imshow(img)
plt.show()
```

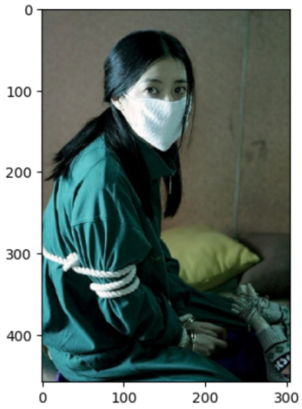

Figure 5: Image of person with mask plotted from the dataset

```
# no mask image
img = mpimg.imread(train_data_path+'/no_mask/'+'train_3517.jpg')
imgplot = plt.imshow(img)
plt.show()
```

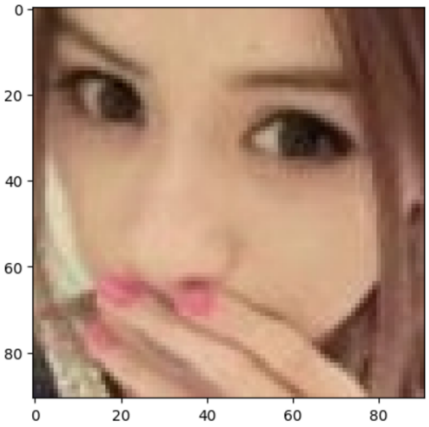

Figure 6: Image of person without mask plotted from the dataset

1.5 Data Preparation

After confirming that the data from the dataset can be loaded into the notebook, the next step is to prepare the data for building the model. Using the ImageDataGenerator from Tensorflow library, it helps to resize all the images in the dataset to the same size of 150 by 150. The dataset is loaded with a batch size of 16 and a binary class mode. From this preparation, the results return 2 classes from each of the train and test data frame, which confirms the categories that was created.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train = ImageDataGenerator(rescale=1/255)
test = ImageDataGenerator(rescale=1/255)
```

```
train_df = train.flow_from_directory(train_data_path,
                                     target_size=(150,150),
                                     batch_size=16,
                                     class_mode='binary')

test_df = test.flow_from_directory(test_data_path,
                                     target_size=(150,150),
                                     batch_size=16,
                                     class_mode='binary')
```

```
Found 313 images belonging to 2 classes.
Found 161 images belonging to 2 classes.
```

Figure 7: Data Preparation for model building

1.6 Convolutional Neural Network (CNN) Model

1.6.1 Model Building

Model building was the next step after preparation of data. Using the CNN model from the TensorFlow Keras library, I build the model with the parameters as shown in figure 8. The model takes the 3D image and go through different convolutional layers which are used for feature extraction. It is then flatten and dense to give a binary result of [0, 1]. The model can be outputted as a summary as shown in figure 9 and it shows that this CNN model have 944,393 trainable parameters. The high number of parameters can be a good thing, as it can help in learning more complex data and having the capacity to fit larger datasets. However, there are some drawbacks, such as longer training time taken for the model.

```
model=Sequential()
model.add(Conv2D(32,(3,3),activation='relu',input_shape=(150,150,3)))
model.add(MaxPooling2D() )
model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPooling2D() )
model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPooling2D() )
model.add(Flatten())
model.add(Dense(100,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
```

Figure 8: Building CNN model

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 32)	9,248
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 32)	0
conv2d_2 (Conv2D)	(None, 34, 34, 32)	9,248
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 32)	0
flatten (Flatten)	(None, 9248)	0
dense (Dense)	(None, 100)	924,900
dense_1 (Dense)	(None, 1)	101

Total params: 944,393 (3.60 MB)
Trainable params: 944,393 (3.60 MB)
Non-trainable params: 0 (0.00 B)

Figure 9: CNN model summary

1.6.2 Fitting the model

After setting up the model, I proceed to fit the model with the train and test dataset. The test dataset was relabelled as validation dataset, so that any observation on the accuracy and loss can be observed over the epochs. For this model, I have set the number of epochs with the value of 10, this means the model was trained 10 times. After the model was trained, analysis and evaluation can be done.

1.6.3 Model Observation and Evaluation

Figure 10 shows the values for the accuracy and losses on the train and validation dataset over each epochs. I went on to plot it on a line graph to show the changes in the values over each epochs as seen in figure 11 and 12.

```
#fitting the model

cnn_model = model.fit(train_df, validation_data=test_df, epochs=10)
```

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyD
d call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queu
s these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
20/20 ━━━━━━━━━━━━━━━━━━━━ 108s 4s/step - accuracy: 0.5653 - loss: 0.7550 - val_accuracy: 0.7950 - val_loss: 0.3959
Epoch 2/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 14s 650ms/step - accuracy: 0.8078 - loss: 0.4298 - val_accuracy: 0.7950 - val_loss: 0.4873
Epoch 3/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 17s 812ms/step - accuracy: 0.8686 - loss: 0.3339 - val_accuracy: 0.8012 - val_loss: 0.3782
Epoch 4/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 16s 749ms/step - accuracy: 0.9010 - loss: 0.2636 - val_accuracy: 0.8696 - val_loss: 0.3451
Epoch 5/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 14s 665ms/step - accuracy: 0.9399 - loss: 0.1584 - val_accuracy: 0.8634 - val_loss: 0.3544
Epoch 6/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 12s 591ms/step - accuracy: 0.9168 - loss: 0.1917 - val_accuracy: 0.8199 - val_loss: 0.4984
Epoch 7/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 12s 579ms/step - accuracy: 0.9788 - loss: 0.0870 - val_accuracy: 0.7764 - val_loss: 0.7876
Epoch 8/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 14s 652ms/step - accuracy: 0.9466 - loss: 0.1382 - val_accuracy: 0.8012 - val_loss: 0.4479
Epoch 9/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 19s 575ms/step - accuracy: 0.9874 - loss: 0.0761 - val_accuracy: 0.8447 - val_loss: 0.6069
Epoch 10/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 12s 575ms/step - accuracy: 0.9742 - loss: 0.0766 - val_accuracy: 0.8882 - val_loss: 0.4824
```

Figure 10: Model accuracy values

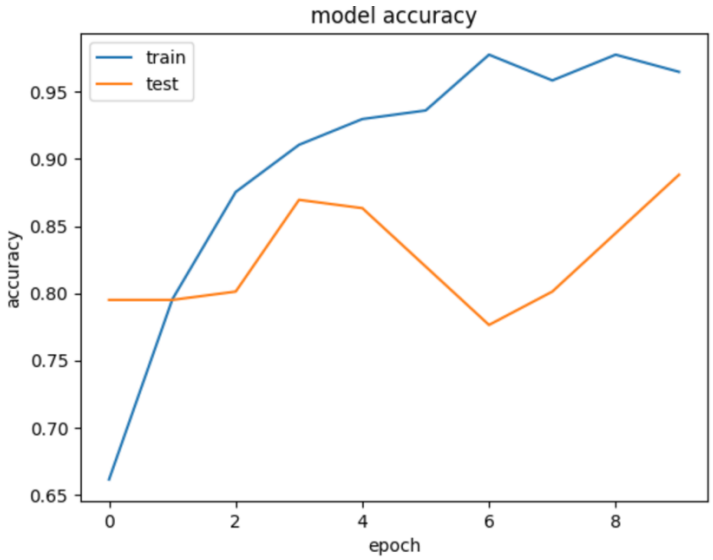

Figure 11: Model Accuracy over Epochs

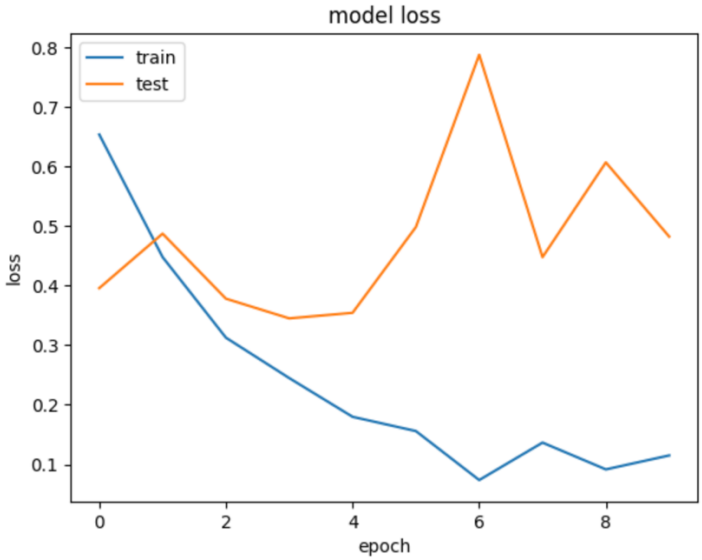

Figure 12: Model Loss over Epochs

From the above figures 10 to 12, I can see that accuracy of the model for the train dataset increases over each epochs from 0.5653 to 0.9742 accuracy. The model loss also decreases from 0.7550 to 0.0766. Although the test dataset have an increase in accuracy from 0.7950 to 0.8882, at the 6 epoch, the accuracy drops below 0.8 before returning above 0.8 accuracy. The model loss are also higher for the test dataset compared to the train dataset with values from 0.3959 to 0.4824 over the 10 epochs. At the 6 epoch, where the accuracy dipped, the loss value also increases to its peak giving the value of 0.7876. Given the higher value for the test dataset, it could mean that the model was overfitted for the test dataset as there might be new classes that are not defined. However the model still provides a 88.82% accuracy as shown in figure 13. This means that the model is still relatively accurate in detecting whether a person is wearing a mask.

```
# evaluate the model

model_evaluate = model.evaluate(test_df)
print(f"Model Accuracy: {model_evaluate[1] * 100 : 0.2f} %")
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 153ms/step - accuracy: 0.9060 - loss: 0.4184
Model Accuracy:  88.82 %
```

Figure 13: Model accuracy

1.7 Model Prediction and testing

The final step is to test the model if it is working on images that were not in the dataset. The images was uploaded into the model for it to predict whether the person is wearing a mask as shown in figure 14. The third and fourth images of figure 14 shows the person does have a face mask on but it is not worn properly. I also used this images to see if the model would be able to detect the presence of a face mask.

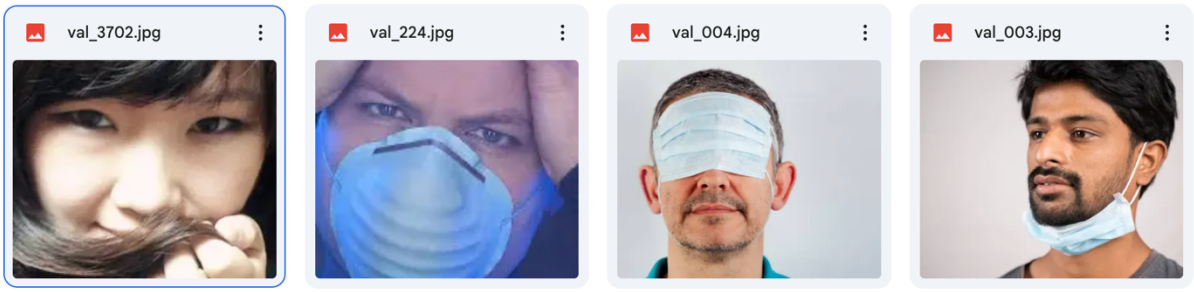

Figure 14: Test images for model prediction

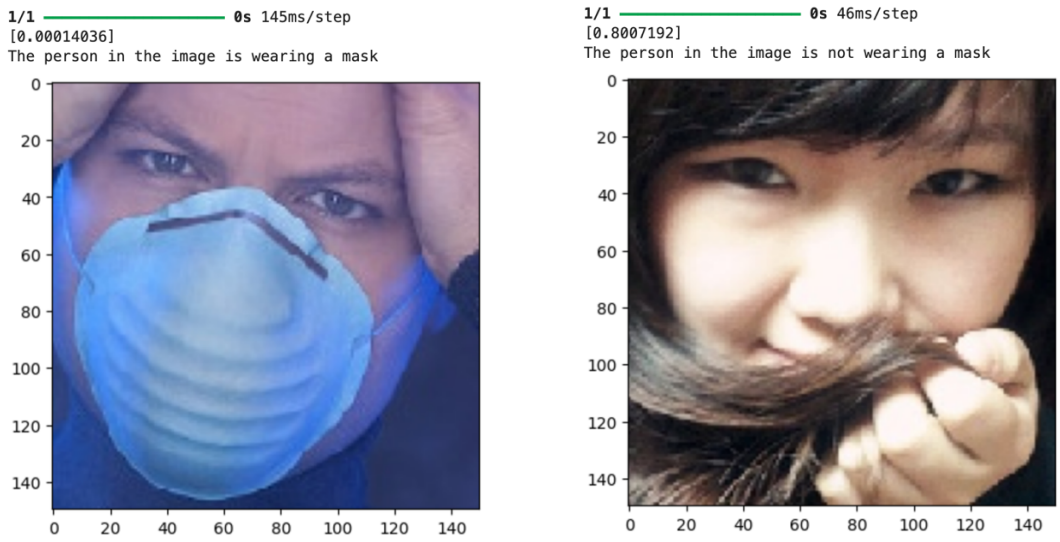

Figure 15: Prediction results from the model

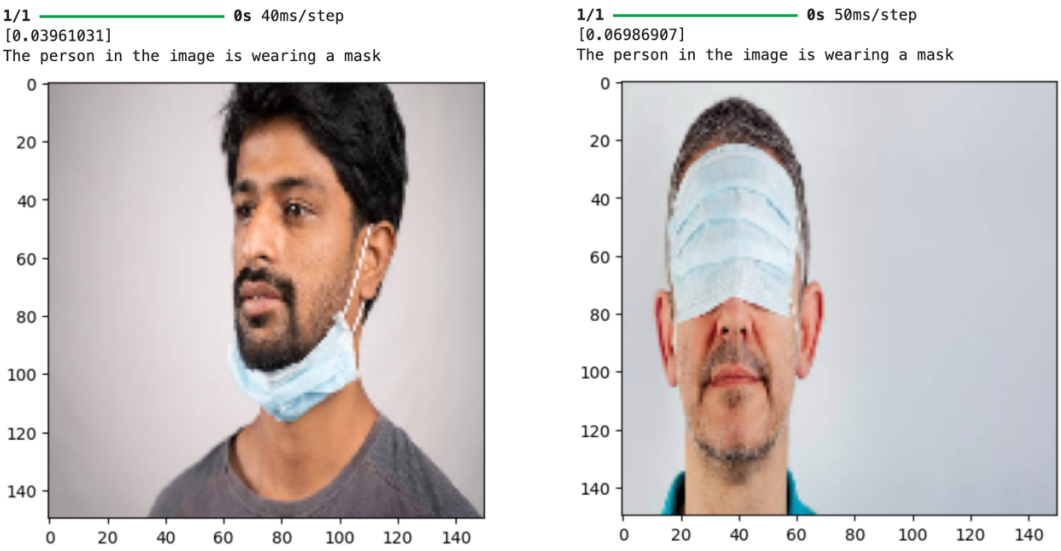

Figure 16: Prediction results from the model

Figure 15 shows that the model was able to correctly predict the whether the person is wearing a mask. It is also able to confidently predict the person that is not wearing a mask, with the confidence level of 0.8. However the confidence level for predicting person wearing a mask is lower with a range of 0.0001 to 0.06 among the 3 images. The model was also able to predict the person wearing a mask in figure 16 even though they are both wearing it incorrectly. Further fine tuning can be performed on the model to train it to detect the proper way of wearing a mask and improve its confidence level.

1.8 Conclusion

In conclusion, the CNN model was able to give accurate results on the detection of a mask, which shows that it can be a useful algorithm tool to use for image processing. The CNN model that was built for this project is just the foundation on mask detection. Improvements such as adding additional classes to detect the proper mask wearing or including the incorrect way of mask wearing in the no mask folder of the training dataset so to improve the model accuracy. Future development such as additional real-time analysing and prediction can be added, so that it can be used to perform real-time detection.

1.9 References

Tampus, C. S. (2020, May 20). How Covid-19 brought South-east Asia's megacities to a standstill | The Straits Times. *The Straits Times*. https://www.straitstimes.com/multimedia/graphics/2020/05/covid-asia-standstill/index.html?shell

COVID-19 epidemiological update – 24 December 2024. (2025, January 23). https://www.who.int/publications/m/item/covid-19-epidemiological-update---24-december-2024

Arm Ltd. (n.d.). *What is a convolutional neural network (CNN)?* Arm | the Architecture for the Digital World. https://www.arm.com/glossary/convolutional-neural-network