How Can Computer Vision Help The Authorities?

Face Mask Detection in The Era of The COVID-19

Rio Pramana
Computer Science Department
School of Computer Science
Bina Nusantara University
Jakarta, Indonesia 11480
rio.pramana@binus.ac.id

Debora
Computer Science Department
School of Computer Science
Bina Nusantara University
Jakarta, Indonesia 11480
debora002@binus.ac.id

Diaz D. Santika
Computer Science Department
School of Computer Science
Bina Nusantara University
Jakarta, Indonesia 11480
ddsantika@binus.ac.idl

***Abstract*—CoronaVirus Disease (COVID-19) pandemic is causing a health crisis in every region in the world, especially in Indonesia. One of the effective methods against the virus is wearing face masks in public places. This paper introduces a new and enhanced face mask recognizer using computer vision and image processing techniques. The face mask recognition in this study is developed with a computer vision algorithm. For the purpose of this study, 7 newly trained convolutional neural networks were used and compared. The performance of the given models were assessed. The final model presented in this paper reached a 100% accuracy.**

Keywords—Face Mask Detection, Computer Vision, COVID-19, Haar cascade classifier, simple CNN

I. Introduction

Since the declaration of the COVID-19 virus as a pandemic by the World Health Organization (WHO), lots of efforts have been made in order to reduce the risk of spreading. Preventing the spread of virus in an efficient way is by using a mask [1]. The face mask should also be identifiable from all sides. Not only is it crucial to wear a mask, but it must also be worn such that the entire mouth and nose are covered. Inappropriate mask use not only increases the risk of viral transmission but also offers little protection [2].

Masks are a crucial factor to stop the transmission of viruses. Due to this, a face mask recognizer is now required, one that can accurately determine who is wearing a face mask in addition to merely detecting its presence can help prevent the outburst of the virus [3]. The main objective is to identify whether the person displayed on the image is wearing a face mask or not with the help of computer vision and image processing. In this study, a face mask was detected using the Convolutional Neural Network (CNN) method. Identifying people that have a mask or have no mask on is a crucial task in today's world.

II. Methodology

There are a few major steps taken in the methodology of this experiment as shown in the figure below. We keep reiterating the model that we created until we found some models that are good enough to be looked into further.

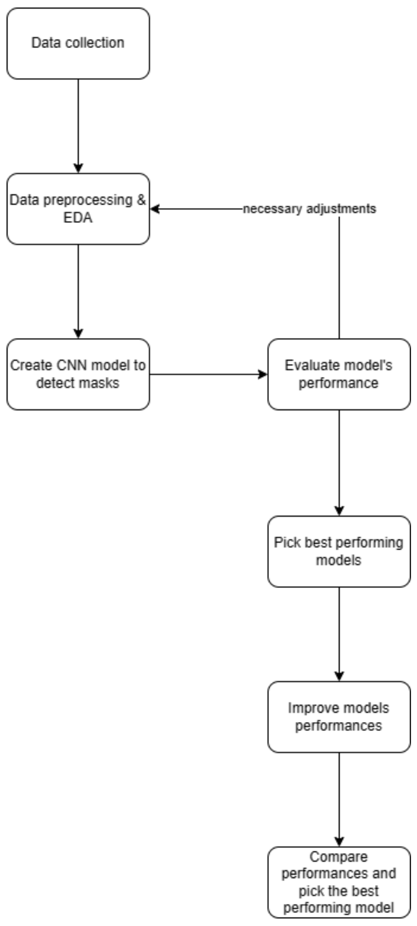

Data collection
Data preprocessing & EDA
necessary adjustments
Create CNN model to detect masks
Evaluate model's performance
Pick best performing models
Improve models performances
Compare performances and pick the best performing model

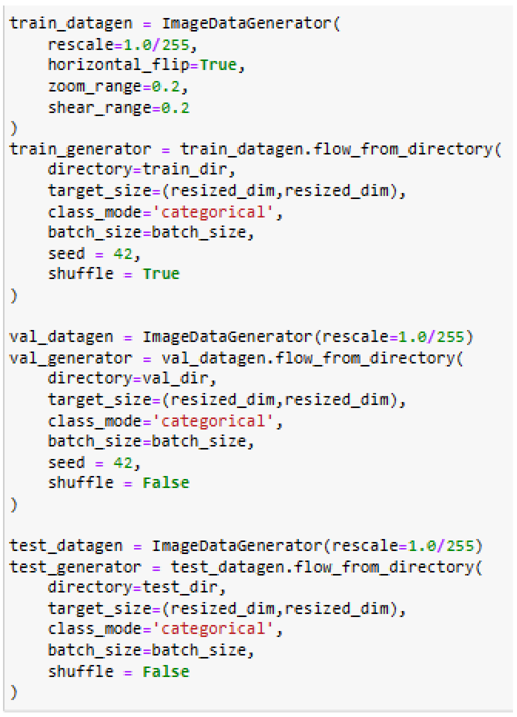
Fig. 1. Experiment methodology

A. Data Collection

For this study, we collected the data from an available Kaggle dataset. The datasets contain images of people with and without masks. The first dataset (https://www.kaggle.com/datasets/ashishjangra27/face-mask-12k-images-dataset) is used as the train and validation dataset. The second dataset (https://www.kaggle.com/datasets/vijaykumar1799/face-mask-detection) is used for the test dataset. This dataset is divided into two classes, which are 'WithMask' and 'WithoutMask'. This dataset is convenient to be used because all of the images have been cropped. In which, it does not require face detection anymore as the images have been cropped into the appropriate faces. We also collected real life samples of images of people using masks and without using masks. The images can be directly trained into the model.

B. Preprocessing Data and Exploratory Data Analysis

The dataset that we are using for testing and training our model is from an open datasource on kaggle that contains images with and without facemasks. In order to efficiently use the dataset, we split the dataset into 80/10/10. There are a total of 10.000 training datasets (composed of 5.000 with mask and 5.000 without mask), 1.250 validation datasets (composed of 625 with mask and 625 without mask) and 1.250 test datasets (composed of 625 with mask and 625 without mask). As we have collected the data, then it is loaded to perform pre-processing. Dataset preprocessing is an important step to build projects using a set of data. There are various data pre-processing techniques that we used. The data is split into a training, validation, and testing set. Training dataset will be used to train the model while testing data will be used to test the model. A fully trained and tested model results in effective and accurate detection of the presence or absence of a face mask. We resize the image size, this helps in reducing computational requirements and the algorithm is simplified. We also performed data augmentation to increase the training performance of the data and resulted in an effective model.



```
train_datagen = ImageDataGenerator(
    rescale=1.0/255,
    horizontal_flip=True,
    zoom_range=0.2,
    shear_range=0.2
)
train_generator = train_datagen.flow_from_directory(
    directory=train_dir,
    target_size=(resized_dim,resized_dim),
    class_mode='categorical',
    batch_size=batch_size,
    seed = 42,
    shuffle = True
)

val_datagen = ImageDataGenerator(rescale=1.0/255)
val_generator = val_datagen.flow_from_directory(
    directory=val_dir,
    target_size=(resized_dim,resized_dim),
    class_mode='categorical',
    batch_size=batch_size,
    seed = 42,
    shuffle = False
)

test_datagen = ImageDataGenerator(rescale=1.0/255)
test_generator = test_datagen.flow_from_directory(
    directory=test_dir,
    target_size=(resized_dim,resized_dim),
    class_mode='categorical',
    batch_size=batch_size,
    shuffle = False
)
```

Fig. 2. Dataset augmentation

The seed is set to 42 as a way to retain consistency when we are re-training the model. Shuffling the data affects the learning process, we only shuffle the training data because the model is not learning when it is in validation and testing mode [4].

C. Modeling

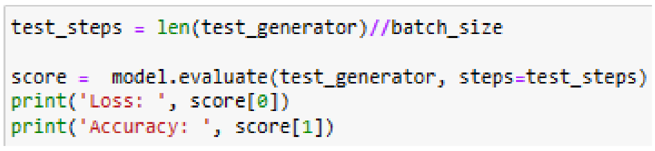
In this study, haarcascade_frontalface_default.xml is used to detect faces and simple CNN is being used to predict whether or not the face has a mask on. However, it is important to note that we do not need to use haarcascade for face detection in the training process of the model as the dataset already contains images that have been cropped according to the face detected. This face detection is only used for the application of the model (receiving new input images from outside of the dataset to be predicted).

Haarcascaade_frontalface_default.xml is a classifier that is composed of a cascade function being trained from a lot of images of both positive and negative, based on this it is used to detect the objects in the other images. The Haar cascade algorithm in detecting the object's face is by training as many as possible positive and negative images and then placing them onto the XML file. A positive image contains an image of a face and a negative image contains an object other than the face [5]. The XML file will speed up detecting the required feature. It uses statistics in detecting faces, by learning to recognize parts of the faces [6].

Simple CNN is one of the important types of neural network that is used for object detection, classify images, image recognition with scene and face recognition. It carries higher feature extraction capabilities and it has a low processing cost, therefore it plays a vital role in face recognition jobs involving computer vision. It uses a convolutional kernel to retrieve high-level features from images [7]. The sequential CNN model that we build has various layers such as Conv2D, MaxPooling2D, Flatten, Dropout and Dense. For the last Dense layer, in most of our models, we use the 'softmax' function that helps in producing a vector output that gives the probability of each of the two classes. 'Adam' optimizer and 'categorical_crossentropy' as our loss function as there are only two classes.

D. Evaluation Metrics

In this study, we are focusing on 'accuracy' as the evaluation metric. We want to focus on building a model that can accurately predict whether or not a person is wearing a mask all the time. There will be 4 different types of accuracy that we will measure on each model's test that will be explained in the next section.



```
test_steps = len(test_generator)//batch_size

score =  model.evaluate(test_generator, steps=test_steps)
print('Loss: ', score[0])
print('Accuracy: ', score[1])
```

Fig. 3. Evaluation method

III. Results & discussion

This section focuses on the results that we found while experimenting to build the best model that we could to solve this problem and discusses the whole process, including decisions and changes that we made that leads us to choosing 1 model that we found gives the best results. We will show and explain the whole experimenting process chronologically.

A. Initial Model

For the architecture of our initial/baseline model, we take reference to the architecture of a CNN model given in BINUS University's 2022/2023 deep learning mid exam.

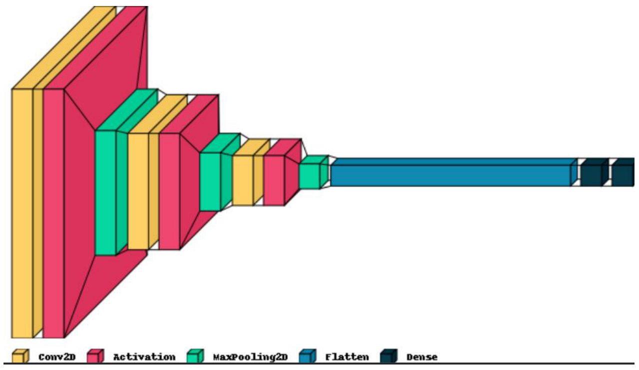
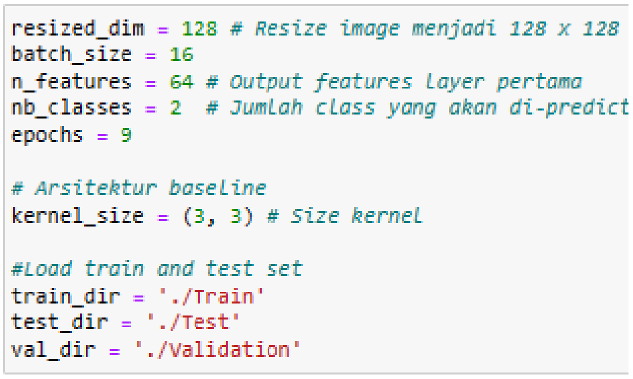


Fig. 4. Baseline model architecture

The requirements for the model are to resize the input image into a 128 x 128 image, where each layer's activation function will be ReLU. The initial kernel size will be 5 x 5, and then it will be 3 x 3 afterwards. The initial output features will be 64 and then it will be halved after each layer.

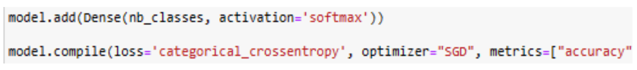


```
resized_dim = 128 # Resize image menjadi 128 x 128
batch_size = 16
n_features = 64 # Output features layer pertama
nb_classes = 2  # Jumlah class yang akan di-predict
epochs = 9

# Arsitektur baseline
kernel_size = (3, 3) # Size kernel

#Load train and test set
train_dir = './Train'
test_dir = './Test'
val_dir = './Validation'
```

Fig. 5. Initial model parameters



```
model.add(Dense(nb_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer="SGD", metrics=["accuracy"
```

Fig. 6. Initial model last layer

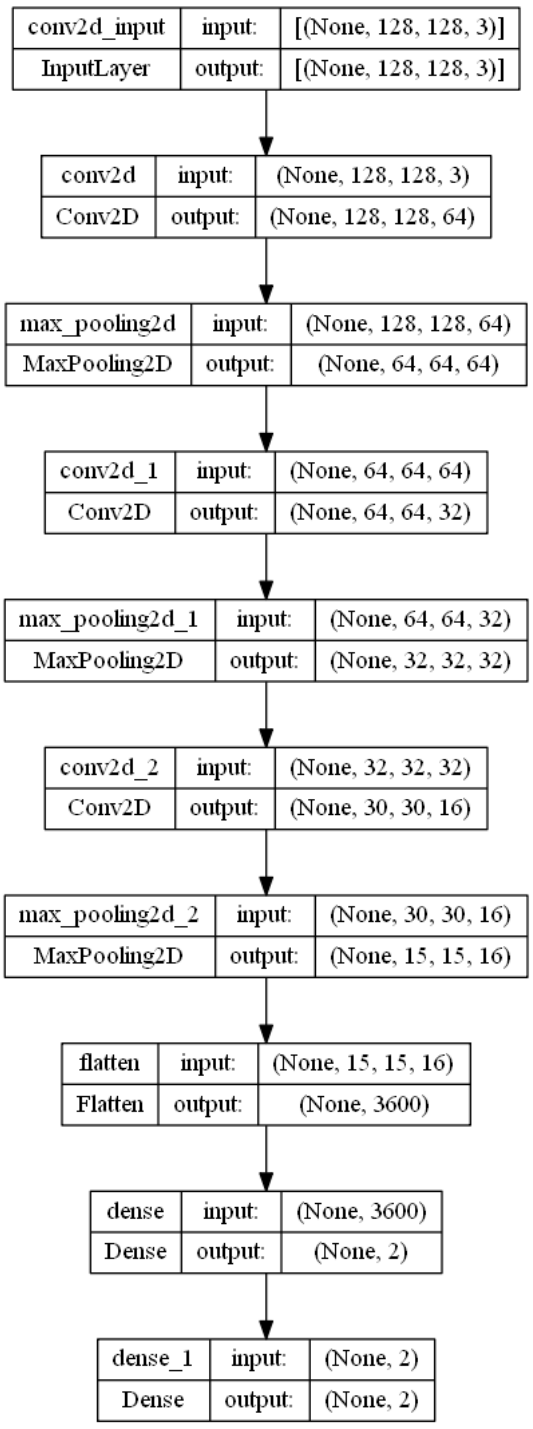
For the batch size, we decided on starting with a batch size of 16 because previous studies have shown that small batch size usually leads to better results [8]. It is believed that a batch size of 16 or 32 is a good starting point [9]. We picked 16 as our starting point because our dataset is not too big (15.000 data).

The number of epochs we are using is 9. We wanted to test several variations of the model before making a decision on which model(s) to focus later on. Therefore, we picked a small number of epochs (9) to make the training process faster while also getting a sufficient amount of training for the model.

For the last layer's activation function, it is believed that 'softmax' and 'sigmoid' are the two best options and perform equally well for binary classification tasks, so we decided to use 'softmax'.

For the loss function, we are using 'categorical_crossentropy' as it is usable for classifications with 2 or more classes. It is believed that 'categorical_crossentropy' is essentially the same as 'binary_crossentropy' when there are only 2 classes like in this case [10].

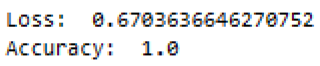
For the optimizer, it is believed that Stochastic Gradient Descent (SGD) and Adam are the two best options for binary classification where both can perform equally well [11]. Therefore, for the baseline model, we tried using SGD first as our optimizer.



conv2d_input	input:	[(None, 128, 128, 3)]
InputLayer	output:	[(None, 128, 128, 3)]
conv2d	input:	(None, 128, 128, 3)
Conv2D	output:	(None, 128, 128, 64)
max_pooling2d	input:	(None, 128, 128, 64)
MaxPooling2D	output:	(None, 64, 64, 64)
conv2d_1	input:	(None, 64, 64, 64)
Conv2D	output:	(None, 64, 64, 32)
max_pooling2d_1	input:	(None, 64, 64, 32)
MaxPooling2D	output:	(None, 32, 32, 32)
conv2d_2	input:	(None, 32, 32, 32)
Conv2D	output:	(None, 30, 30, 16)
max_pooling2d_2	input:	(None, 30, 30, 16)
MaxPooling2D	output:	(None, 15, 15, 16)
flatten	input:	(None, 15, 15, 16)
Flatten	output:	(None, 3600)
dense	input:	(None, 3600)
Dense	output:	(None, 2)
dense_1	input:	(None, 2)
Dense	output:	(None, 2)

Fig. 7. Baseline model

After training the model, using the evaluate() function on the test dataset, this model apparently has an accuracy of 100%.



```
Loss:  0.6703636646270752
Accuracy:  1.0
```

Fig. 8. Baseline model test result

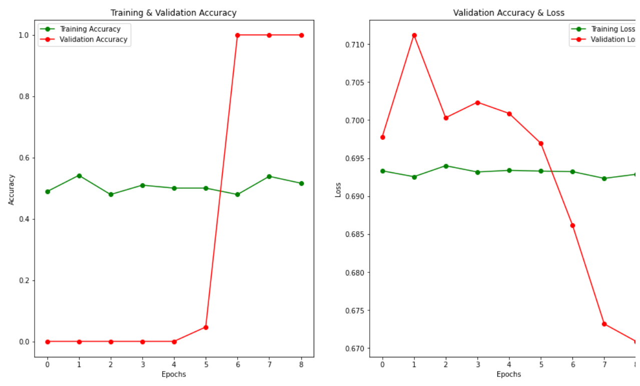


Fig. 9. Baseline model training & validation result graph

However, looking at the model's loss and graph of its training and validation accuracy, we suspected that the accuracy shown in Fig. 9 does not correctly represent the model's capability to detect masks. To confirm our suspicion, we decided to test the model using a demo picture that contains 6 people's faces with half of them wearing a mask.

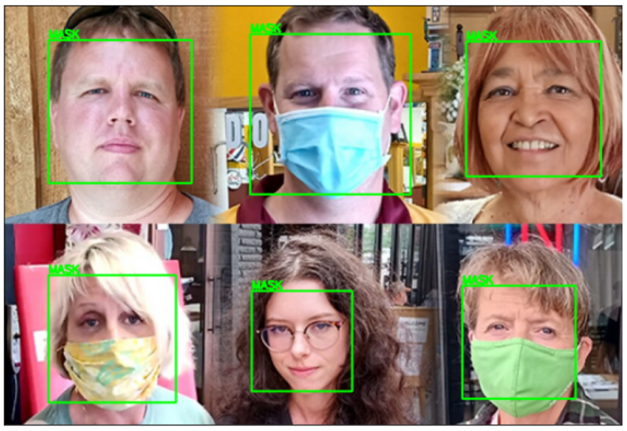
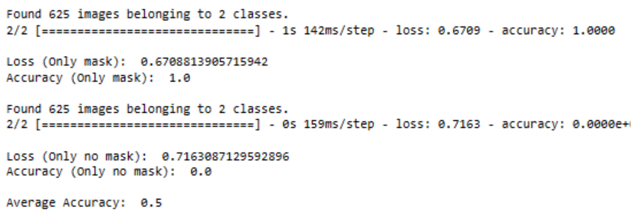


Fig. 10. Baseline model demo result

The demo result shows that the model is not able to make correct predictions, and we suspected the model somehow managed to get 100% accuracy on testing by assuming every single picture it is given is a person wearing a mask (guessing 'mask' every time). To confirm our suspicion again, we made 2 new folders, the first folder is 'Test All Mask' where it only contains pictures of people wearing a mask, while the second folder is 'Test No Mask' where it only contains pictures of people not wearing a mask. If our suspicion is correct, then the model should have a 100% accuracy on the 'Test All Mask' dataset and 0% accuracy on the 'Test No Mask' dataset.



```
Found 625 images belonging to 2 classes.
2/2 [==============================] - 1s 142ms/step - loss: 0.6709 - accuracy: 1.0000

Loss (Only mask):  0.6708813905715942
Accuracy (Only mask):  1.0

Found 625 images belonging to 2 classes.
2/2 [==============================] - 0s 159ms/step - loss: 0.7163 - accuracy: 0.0000e+

Loss (Only no mask):  0.7163087129592896
Accuracy (Only no mask):  0.0

Average Accuracy:  0.5
```

Fig. 11. Baseline model mask & no mask accuracy

It turns out that our suspicion was correct. The model guessed 'WithMask' on every picture it was given. From this point, we decided to keep track of 2 new metrics, 'mask_acc' and 'no_mask_acc' ('avg_acc' is obtained by getting the average value of these 2 metrics) to make it clearer of how the model that we have trained actually performs.

B. *Improving Baseline Model*

To improve the baseline model, we experimented with several variations of the architecture and parameters.

1) *First Model*

From the results of the baseline model, it is clear that the model needs some changes to improve its performance. The first change that we made is the initial kernel size. The baseline model used 5 x 5 kernel size initially, and in this model we will start using 3 x 3 as the kernel size from the very start. We wanted to try keeping the kernel size uniform for the whole model and reducing the training time by using small kernel size. It is also believed that 3 x 3 kernel size might be the optimal size for CNN [12].

The second change that we made is by using Adam as the optimizer. We wanted to see how well the model can perform if we use Adam optimizer instead. We used a learning rate of 0.01 to make the learning process faster as we are only using 9 epochs.

The next change that we made is adding *batch normalization* and *dropout* in between layers. Dropout is applied to minimize the effect of overfitting. The technique has shown that it can improve a model's performance significantly if used on every layer [13]. Adam optimizer is susceptible to overfitting, especially with a high learning rate that we are using [13]. Therefore, we decided to apply the dropout technique with a probability of 0.1.

Batch normalization is used to normalize contributions to a layer for each mini-batch where it results in faster training time and performance improvements [14]. Although it is still debatable, most people believe that putting the batch normalization between the activation function and dropout technique will result in better model performance [15]. Therefore, we decided to apply it that way.

The last change that we made is changing the input to a grayscale image first. We wanted to see how it will affect the model's performance. As it is the first attempt at improving the model, we will call this model 'model_1'.

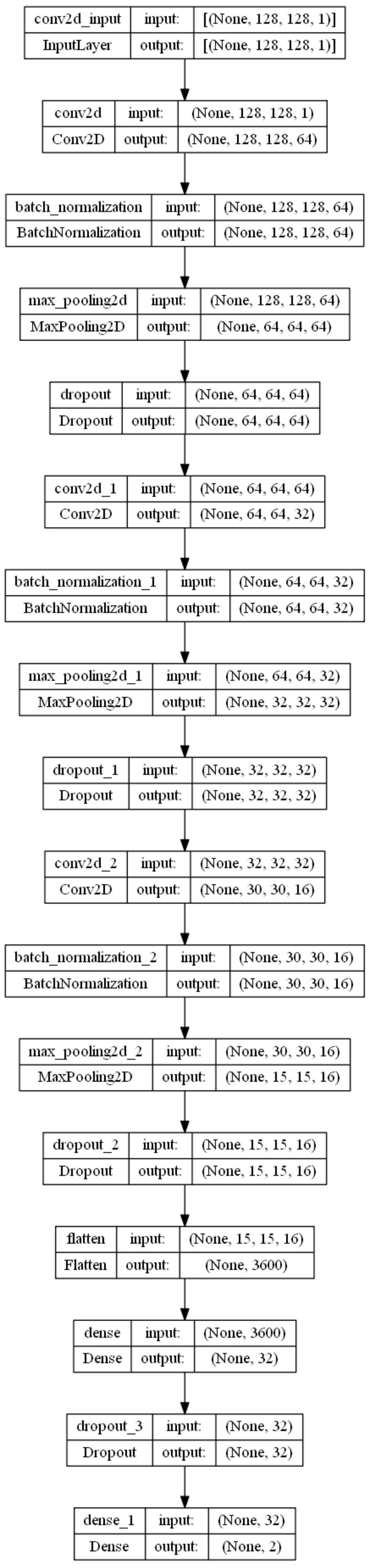

conv2d_input | input: | [(None, 128, 128, 1)]
InputLayer | output: | [(None, 128, 128, 1)]
conv2d | input: | (None, 128, 128, 1)
Conv2D | output: | (None, 128, 128, 64)
batch_normalization | input: | (None, 128, 128, 64)
BatchNormalization | output: | (None, 128, 128, 64)
max_pooling2d | input: | (None, 128, 128, 64)
MaxPooling2D | output: | (None, 64, 64, 64)
dropout | input: | (None, 64, 64, 64)
Dropout | output: | (None, 64, 64, 64)
conv2d_1 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 32)
batch_normalization_1 | input: | (None, 64, 64, 32)
BatchNormalization | output: | (None, 64, 64, 32)
max_pooling2d_1 | input: | (None, 64, 64, 32)
MaxPooling2D | output: | (None, 32, 32, 32)
dropout_1 | input: | (None, 32, 32, 32)
Dropout | output: | (None, 32, 32, 32)
conv2d_2 | input: | (None, 32, 32, 32)
Conv2D | output: | (None, 30, 30, 16)
batch_normalization_2 | input: | (None, 30, 30, 16)
BatchNormalization | output: | (None, 30, 30, 16)
max_pooling2d_2 | input: | (None, 30, 30, 16)
MaxPooling2D | output: | (None, 15, 15, 16)
dropout_2 | input: | (None, 15, 15, 16)
Dropout | output: | (None, 15, 15, 16)
flatten | input: | (None, 15, 15, 16)
Flatten | output: | (None, 3600)
dense | input: | (None, 3600)
Dense | output: | (None, 32)
dropout_3 | input: | (None, 32)
Dropout | output: | (None, 32)
dense_1 | input: | (None, 32)
Dense | output: | (None, 2)

Fig. 12. model_1 architecture

After training the model, the evaluate() function returns an 82.8125% accuracy. The mask_acc is 81.25%, the no_mask_acc is 87.5%, and the avg_acc is 84.375%. Based on the results, we can say that the model has improved now as it can make a prediction on whether a person is wearing a mask or not, instead of always guessing that a person is wearing a mask. However, we want the accuracy to be as high as possible, so we will keep making changes to see if we can improve the model even more.

2) *Second Model*

For 'model_2', we decided to not change the image into a grayscale image, just like the baseline model. In 'model_1', we changed the image into a grayscale image first, but we will not do that in this model. Therefore, the architecture is exactly the same as 'model_1' except for the input image which is slightly different.

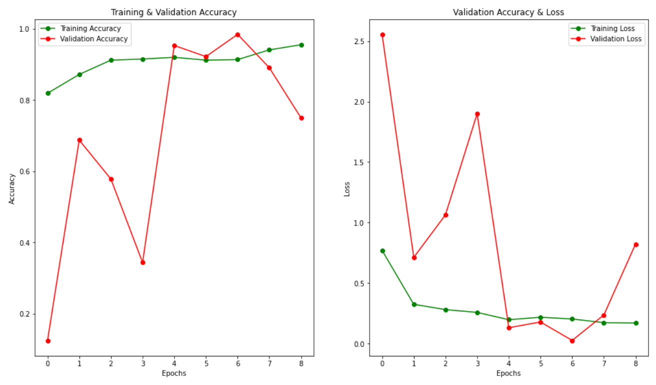


Fig. 13. model_2 training & validation results graph

'model_2' managed to see significant improvement in accuracy. Its accuracy is 95.3125%, mask_acc is 93.75%, no_mask_acc is 100%, and avg_acc is 96.875%. It is almost perfect in terms of the accuracy numbers, and when we tested it on the demo picture, it managed to make correct predictions.

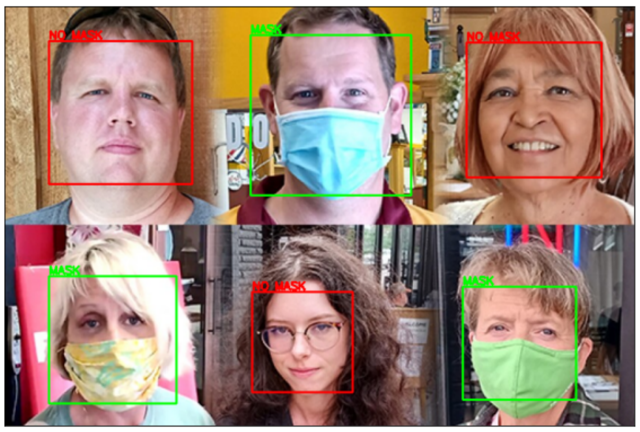


Fig. 14. model_2 demo result

However, we still wanted to try other variations as we want to try to reach the highest accuracy possible. We also saw a problem in the graph where the model seems to start overfitting on epochs 8 & 9, but we will fix it in the later section of this paper.

3) *Third Model*

For 'model_3', we decided to try adding a fourth convolutional layer. We wanted to see if a more complex model will result in a better performance. The additional convolutional layer is exactly the same as the other convolutional layers. We also changed the dropout probability to 0.3 as the architecture has gotten more complex.

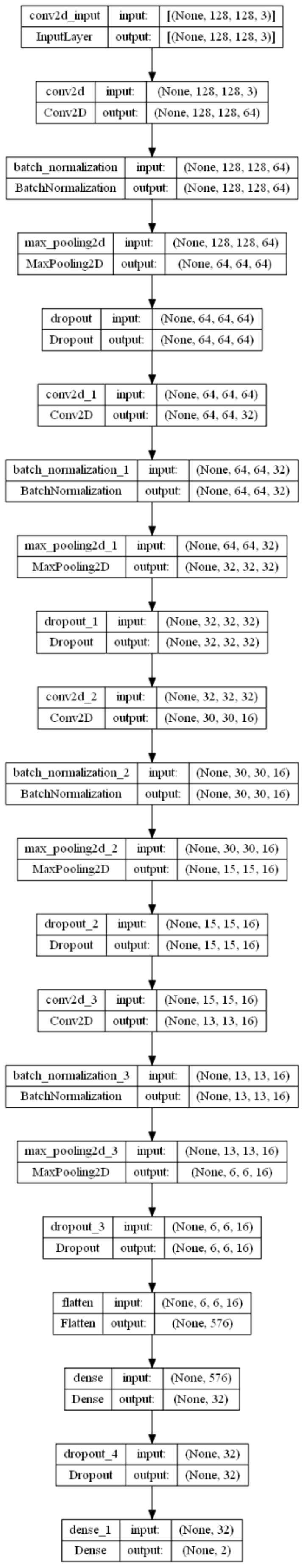

conv2d_input | InputLayer | input: [(None, 128, 128, 3)] | output: [(None, 128, 128, 3)]
conv2d | Conv2D | input: (None, 128, 128, 3) | output: (None, 128, 128, 64)
batch_normalization | BatchNormalization | input: (None, 128, 128, 64) | output: (None, 128, 128, 64)
max_pooling2d | MaxPooling2D | input: (None, 128, 128, 64) | output: (None, 64, 64, 64)
dropout | Dropout | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
conv2d_1 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 32)
batch_normalization_1 | BatchNormalization | input: (None, 64, 64, 32) | output: (None, 64, 64, 32)
max_pooling2d_1 | MaxPooling2D | input: (None, 64, 64, 32) | output: (None, 32, 32, 32)
dropout_1 | Dropout | input: (None, 32, 32, 32) | output: (None, 32, 32, 32)
conv2d_2 | Conv2D | input: (None, 32, 32, 32) | output: (None, 30, 30, 16)
batch_normalization_2 | BatchNormalization | input: (None, 30, 30, 16) | output: (None, 30, 30, 16)
max_pooling2d_2 | MaxPooling2D | input: (None, 30, 30, 16) | output: (None, 15, 15, 16)
dropout_2 | Dropout | input: (None, 15, 15, 16) | output: (None, 15, 15, 16)
conv2d_3 | Conv2D | input: (None, 15, 15, 16) | output: (None, 13, 13, 16)
batch_normalization_3 | BatchNormalization | input: (None, 13, 13, 16) | output: (None, 13, 13, 16)
max_pooling2d_3 | MaxPooling2D | input: (None, 13, 13, 16) | output: (None, 6, 6, 16)
dropout_3 | Dropout | input: (None, 6, 6, 16) | output: (None, 6, 6, 16)
flatten | Flatten | input: (None, 6, 6, 16) | output: (None, 576)
dense | Dense | input: (None, 576) | output: (None, 32)
dropout_4 | Dropout | input: (None, 32) | output: (None, 32)
dense_1 | Dense | input: (None, 32) | output: (None, 2)

Fig. 15. model_3 architecture

After training the model, the results show that the performance of this model is worse than model_2. It was able to get a 100% mask_acc but only 46.875% no_mask_acc.

4) *Fourth Model*

From the results of model_3 and looking at its architecture, we figured that the model could probably benefit from having a bigger input image dimensions as it has an additional convolutional layer which will affect the total number of output features in the end. Therefore, for model_4, we decided to try changing the input image dimensions to 256 (the input image will have a size of 256 x 256) while the rest of the architecture and parameters remain the same.

After training the model, it managed to get 100% mask_acc and 93.75% no_mask_acc. This model achieved the same performance as our highest performing model yet (model_2). However, the model also faced the same problem as model_2 which is overfitting.

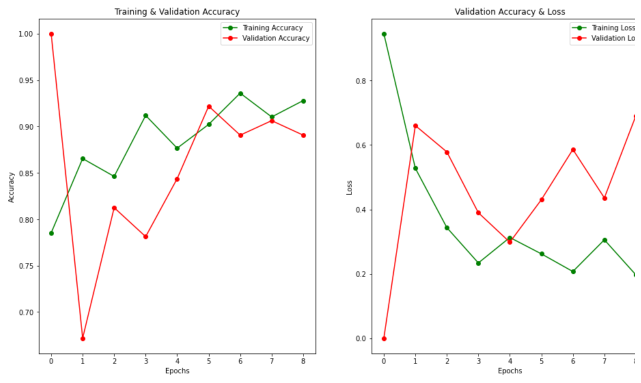
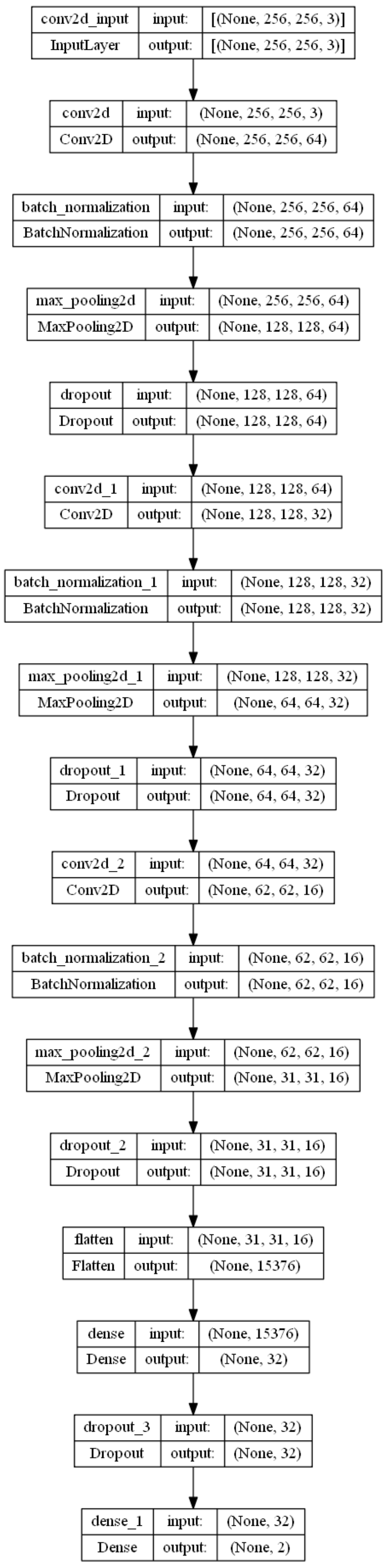


Fig. 16. model_4 training & validation results graph

5) *Fifth Model*

After the success of model_2 and model_4, we wanted to try going back to model_2's architecture, but keeping the same parameters as model_4 (dropout probability stays at 0.3 and the dimensions of the input image stays at 256 x 256) and see the results.



conv2d_input | input: | [(None, 256, 256, 3)]
InputLayer | output: | [(None, 256, 256, 3)]
conv2d | input: | (None, 256, 256, 3)
Conv2D | output: | (None, 256, 256, 64)
batch_normalization | input: | (None, 256, 256, 64)
BatchNormalization | output: | (None, 256, 256, 64)
max_pooling2d | input: | (None, 256, 256, 64)
MaxPooling2D | output: | (None, 128, 128, 64)
dropout | input: | (None, 128, 128, 64)
Dropout | output: | (None, 128, 128, 64)
conv2d_1 | input: | (None, 128, 128, 64)
Conv2D | output: | (None, 128, 128, 32)
batch_normalization_1 | input: | (None, 128, 128, 32)
BatchNormalization | output: | (None, 128, 128, 32)
max_pooling2d_1 | input: | (None, 128, 128, 32)
MaxPooling2D | output: | (None, 64, 64, 32)
dropout_1 | input: | (None, 64, 64, 32)
Dropout | output: | (None, 64, 64, 32)
conv2d_2 | input: | (None, 64, 64, 32)
Conv2D | output: | (None, 62, 62, 16)
batch_normalization_2 | input: | (None, 62, 62, 16)
BatchNormalization | output: | (None, 62, 62, 16)
max_pooling2d_2 | input: | (None, 62, 62, 16)
MaxPooling2D | output: | (None, 31, 31, 16)
dropout_2 | input: | (None, 31, 31, 16)
Dropout | output: | (None, 31, 31, 16)
flatten | input: | (None, 31, 31, 16)
Flatten | output: | (None, 15376)
dense | input: | (None, 15376)
Dense | output: | (None, 32)
dropout_3 | input: | (None, 32)
Dropout | output: | (None, 32)
dense_1 | input: | (None, 32)
Dense | output: | (None, 2)

Fig. 17. model_5 architecture

After training the model, the results are slightly worse than model_2 and model_4, with mask_acc of 84.375% and no_mask_acc of 93.75%.

6) Sixth Model

We thought that model_5 might see an improvement in performance if we change the dimensions of the input image back to 128 x 128 because the architecture does not have an additional convolutional layer like model_4 had. Therefore, we decided to run another test with this architecture, but we only changed the input image size back to 128 x 128.

After training the model, the performance has improved from model_5 but it did not reach the level of performance model_2 and model_4 had. It has a mask_acc of 84.375% and no_mask_acc of 100%. model_6 also has a problem with overfitting which will be solved in the next section.

C. Maximizing the Best Models

After training the baseline model and 6 other models, we picked the best performing models to train further.

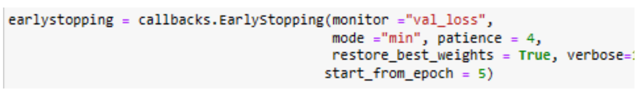
TABLE I. Model Training Results

Model Number	**Accuracy (%)**	**mask_acc (%)**	**no_mask_ acc (%)**	**avg_acc (%)**
Baseline	100	100	0	50
1	82.8125	81.25	87.5	84.375
2	95.3125	93.75	100	96.875
3	98.4375	100	46.875	73.4375
4	98.4375	100	93.75	96.875
5	87.5	84.375	93.75	89.0625
6	85.9375	84.375	100	92.1875

The table above summarizes the results of the 7 models that we trained where 'Accuracy' means the accuracy obtained using evaluate() method and 'avg_acc' means the sum of 'mask_acc' and 'no_mask_acc' divided by 2.

There are 4 models that we think stand out when looking at the 'Accuracy' and 'avg_acc', they are model number 2, 4, 5, and 6. Therefore, we will be focusing on these 4 models and try to maximize their performance.

We noticed that all 4 of these models have a common problem, which is overfitting even though the number of epochs is already small (9). To maximize their performance, we tried to train these 4 models with as many epochs as they need while avoiding overfitting the models. To achieve this, we implemented EarlyStopping from Keras to monitor and end the model's training when it starts to show signs of overfitting.



```
earlystopping = callbacks.EarlyStopping(monitor ="val_loss",
                                        mode ="min", patience = 4,
                                        restore_best_weights = True, verbose=
                                       start_from_epoch = 5)
```

Fig. 18. EarlyStopping implementation

What we are trying to achieve is training the model with as many epochs as it needs, but stop the training once it starts overfitting. Therefore, we will be monitoring the 'val_loss' with mode 'min' as we are evaluating the validation loss and we are trying to minimize the validation loss as much as possible. We decided to put the 'patience' value to 4 as we are still using a relatively high learning rate (0.01), so if the validation loss shows no improvement in 4 epochs, then we can be sure that it is probably the best time to stop the training.

We have restore_best_weights on 'True' because we want to be using the model weights from the epoch that returns the best performance of the model. We also set start_from_epoch to '5' as we noticed that sometimes the validation loss peaked at the first few epochs (while the training loss is still huge), so we decided to monitor the validation loss from epoch 5 to make sure we can properly train and evaluate the model.

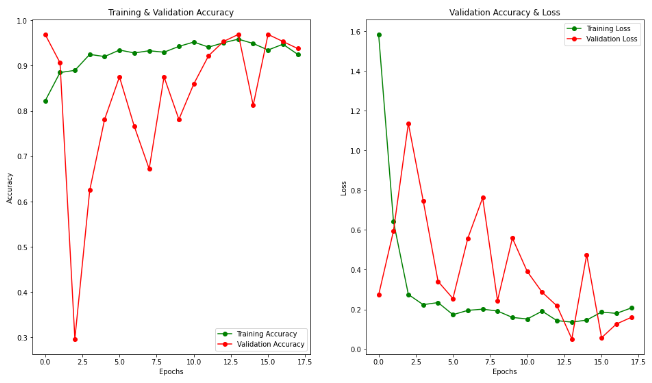


Fig. 19. model_2 training & validation results graph

After training the 4 picked models with maximum epochs of 40, we managed to get the proper weights of each model that avoids overfitting. All models found the best validation loss without overfitting before 20 epochs.

TABLE II. Picked Model Re-training Results

Model Number	**Accuracy (%)**	**mask_acc (%)**	**no_mask_ acc (%)**	**avg_acc (%)**
2	100	100	68.75	84.375
4	98.4375	100	59.375	79.6875
5	96.875	100	84.375	92.1875
6	100	100	100	100

With this method, model number 2, 4, and 5 improved their 'Accuracy' and 'mask_acc', but it seems to worsen their 'no_mask_acc'. Despite this, model number 6 actually managed to get 100% accuracy on all metrics.

Because we have found a model that achieved the maximum accuracy, we decided to stop the experiment and use model number 6 as our solution.

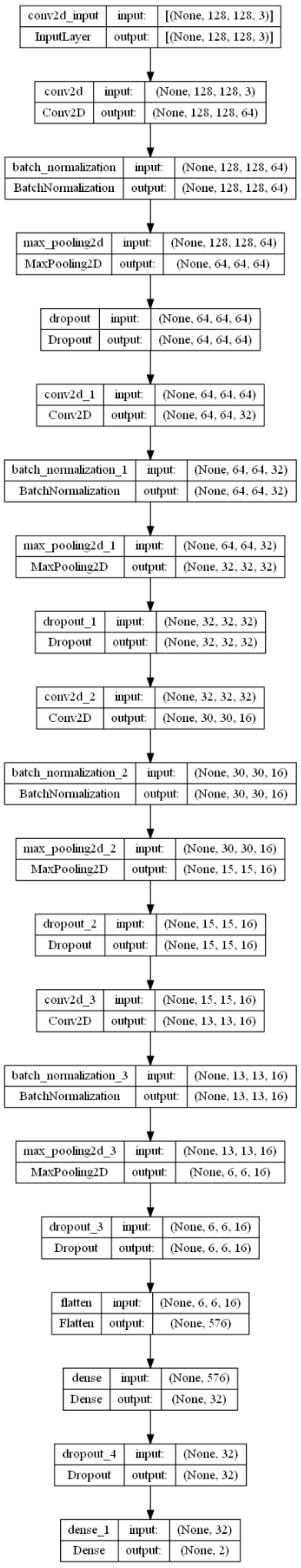
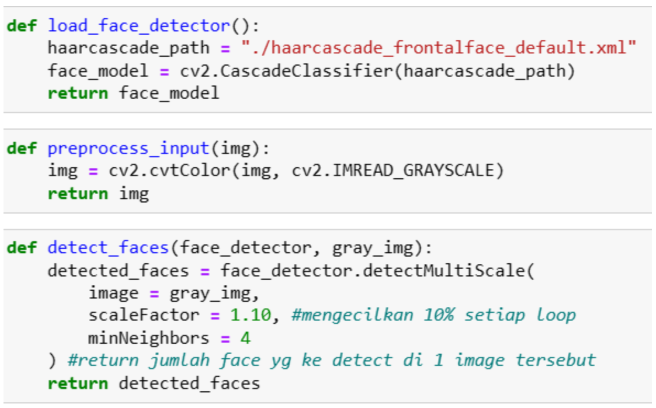

conv2d_input | InputLayer | input: [(None, 128, 128, 3)] | output: [(None, 128, 128, 3)]
conv2d | Conv2D | input: (None, 128, 128, 3) | output: (None, 128, 128, 64)
batch_normalization | BatchNormalization | input: (None, 128, 128, 64) | output: (None, 128, 128, 64)
max_pooling2d | MaxPooling2D | input: (None, 128, 128, 64) | output: (None, 64, 64, 64)
dropout | Dropout | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
conv2d_1 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 32)
batch_normalization_1 | BatchNormalization | input: (None, 64, 64, 32) | output: (None, 64, 64, 32)
max_pooling2d_1 | MaxPooling2D | input: (None, 64, 64, 32) | output: (None, 32, 32, 32)
dropout_1 | Dropout | input: (None, 32, 32, 32) | output: (None, 32, 32, 32)
conv2d_2 | Conv2D | input: (None, 32, 32, 32) | output: (None, 30, 30, 16)
batch_normalization_2 | BatchNormalization | input: (None, 30, 30, 16) | output: (None, 30, 30, 16)
max_pooling2d_2 | MaxPooling2D | input: (None, 30, 30, 16) | output: (None, 15, 15, 16)
dropout_2 | Dropout | input: (None, 15, 15, 16) | output: (None, 15, 15, 16)
conv2d_3 | Conv2D | input: (None, 15, 15, 16) | output: (None, 13, 13, 16)
batch_normalization_3 | BatchNormalization | input: (None, 13, 13, 16) | output: (None, 13, 13, 16)
max_pooling2d_3 | MaxPooling2D | input: (None, 13, 13, 16) | output: (None, 6, 6, 16)
dropout_3 | Dropout | input: (None, 6, 6, 16) | output: (None, 6, 6, 16)
flatten | Flatten | input: (None, 6, 6, 16) | output: (None, 576)
dense | Dense | input: (None, 576) | output: (None, 32)
dropout_4 | Dropout | input: (None, 32) | output: (None, 32)
dense_1 | Dense | input: (None, 32) | output: (None, 2)

Fig. 20. model_6 architecture

D. Demo Results

Using the improved model_6, we tested the model on several demo pictures to check its capability to detect masks. The demo pictures that we used are real images, meaning that it has not been cropped to only show their face(s). Therefore, before passing the input image into the model, we first have to perform face detection using a haar cascade.



```
def load_face_detector():
    haarcascade_path = "./haarcascade_frontalface_default.xml"
    face_model = cv2.CascadeClassifier(haarcascade_path)
    return face_model
```

```
def preprocess_input(img):
    img = cv2.cvtColor(img, cv2.IMREAD_GRAYSCALE)
    return img
```

```
def detect_faces(face_detector, gray_img):
    detected_faces = face_detector.detectMultiScale(
        image = gray_img,
        scaleFactor = 1.10, #mengecilkan 10% setiap loop
        minNeighbors = 4
    ) #return jumlah face yg ke detect di 1 image tersebut
    return detected_faces
```

Fig. 21. Haar cascade face detection for demo pictures

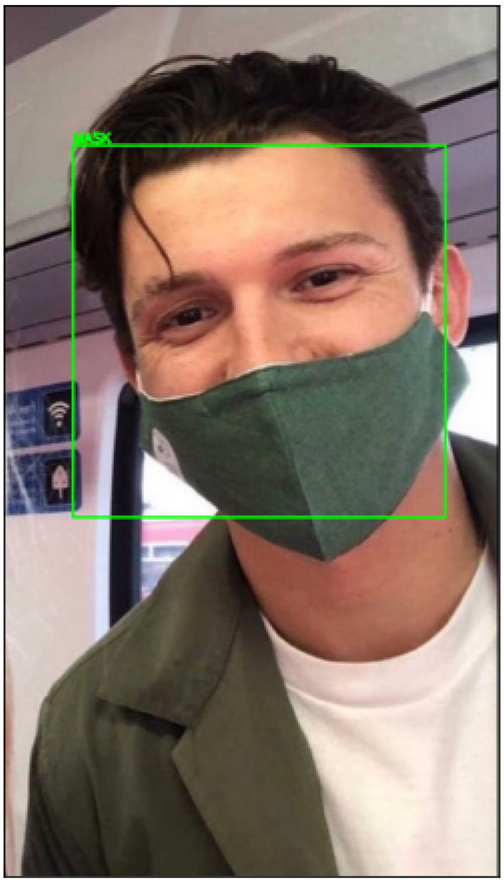


Fig. 22. First demo result

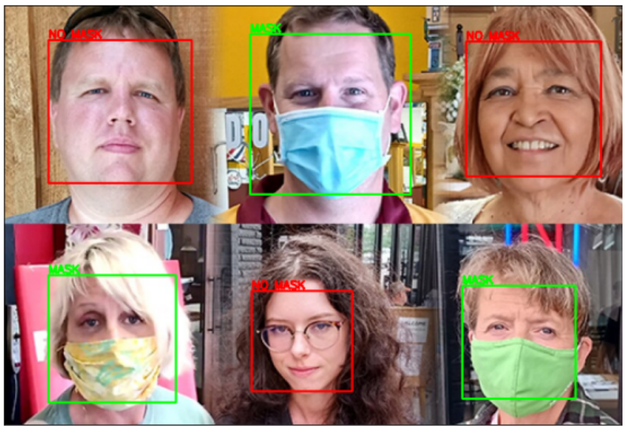


Fig. 23. Second demo result

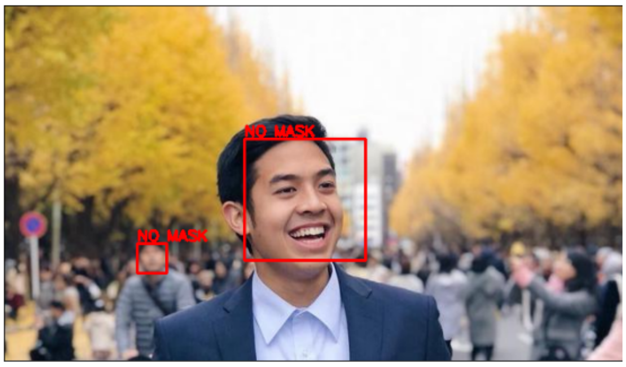


Fig. 24. Third demo result

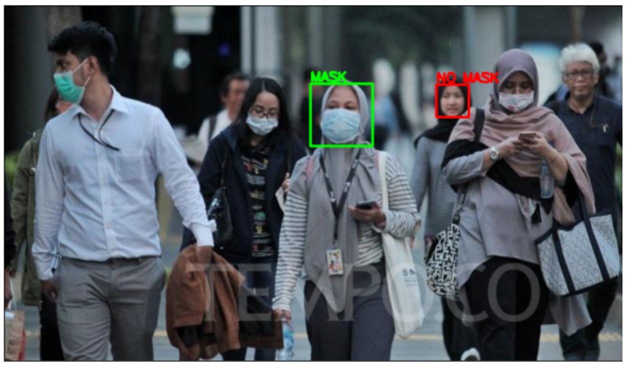


Fig. 25. Fourth demo result

From the demo results, the model is able to correctly classify whether a detected face is wearing a mask or not. However, the application depends on whether or not a face is detected, so when there is one or more undetected faces, the model has no way of classifying whether the person whose face is undetected is wearing a mask or not.

IV. Conclusion

This paper tried to combine haar cascade face detection with a CNN model that can detect masks on the detected faces. It is found that the architecture of the final model using max pooling, batch normalization, dropout technique, along with its parameters was able to reach 100% accuracy and correctly predict masks on detected faces. We believe this model can be a good foundation/starting point for building a real-time mask detection model that can be used to enforce health mandates involving the use of masks in real life.

However, this experiment did not come without challenges. Therefore, for future works, we suggest looking into several things to further improve the findings of this paper:

1. Using/building a better face detection algorithm so the mask detection model can perform better in real life applications
2. Further improving the model used in this paper by using various different evaluation metrics that might help evaluate the model better
3. Building a real-time mask detection application with the help of the model used in this paper

References

[1] A.S. Joshi, S.S. Joshi, G. Kanahasabai, R. Kapil, S. Gupta, Deep Learning Framework to Detect Face Masks from Video Footage, in: 2020 12th International Conference on Computational Intelligence and Communication Networks (CICN), 2020, pp. 435– 440

[2] A.K. Sharadhi, Vybhavi Gururaj, Sahana P.Shankar, M.S. Supriya, neha Sanjay Chogule (2022). Face mask recogniser using image processing and computer vision approach. Global Transition Proceedings 3 67-73.

[3] A. Alguzo, A. Alzu'bi, F. Albalas, Masked Face Detection using Multi- Graph Convolutional Networks, in: 2021 12th International Conference on Information and Communication Systems (ICICS), 2021, pp. 385–391

[4] *Should we also shuffle the test dataset when training with SGD?* Artificial Intelligence Stack Exchange. (2020, November 3). Retrieved January 22, 2023, from https://ai.stackexchange.com/questions/24391/should-we-also-shuffle-the-test-dataset-when-training-with-sgd#:~:text=There%20is%20no%20point%20to,done%20in%20the%20training%20time.

[5] Radimas Putra M.D.L Sirojul Hadi, Parama Diptya Widayaka, 2020. Low Cost System for Face Mask Detection Based Haar Cascade Classifier Method. Jurnal Manajemen, Teknik Informatika, dan Rekayasa Komputer. Vol. 21, No. 1.

[6] V. K and S.Padmavathi, "Facial Parts Detection Using Viola Jones Algorithm," International Conference on Advanced Computing and Communication Systems (ICACCS -2015), pp. 1–4, 2017.

[7] Md. Anwar Hossain, Md. Shahriar Alam Sajib. 2019. Classification of Image using Convolutional Neural Network (CNN). Global Journal of Computer Science and Technology: D Neural & Artificial Intelligence. Volume 19 Issue 2 Version 1.0.

[8] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836.

[9] Thakur, A. (2020, August 19). *What's the optimal batch size to train a neural network?* W&B. Retrieved January 21, 2023, from https://wandb.ai/ayush-thakur/dl-question-bank/reports/What-s-the-Optimal-Batch-Size-to-Train-a-Neural-Network---VmlldzoyMDkyNDU

[10] *Should I use a categorical cross-entropy or binary cross-entropy loss for binary predictions?* Cross Validated. (2017, February 7). Retrieved January 21, 2023, from https://stats.stackexchange.com/questions/260505/should-i-use-a-categorical-cross-entropy-or-binary-cross-entropy-loss-for-binary

[11] *What kind of optimizer is suggested to use for binary classification of similar images?* Artificial Intelligence Stack Exchange. (2020, February 24). Retrieved January 21, 2023, from https://ai.stackexchange.com/questions/18206/what-kind-of-optimizer-is-suggested-to-use-for-binary-classification-of-similar

[12] Pandey, S. (2020, July 1). *How to choose the size of the convolution filter or kernel size for CNN?* Medium. Retrieved January 21, 2023, from https://medium.com/analytics-vidhya/how-to-choose-the-size-of-the-convolution-filter-or-kernel-size-for-cnn-86a55a1e2d15

[13] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.

[14] Singla, S. (2020, November 23). *Why is batch normalization useful in deep neural network?* Medium. Retrieved January 22, 2023, from https://towardsdatascience.com/batch-normalisation-in-deep-neural-network-ce65dd9e8dbf#:~:text=Batch%20normalization%20solves%20a%20major,use%20a%20higher%20learning%20rate.

[15] *Where do I call the batchnormalization function in Keras?* Stack Overflow. (2016, January 11). Retrieved January 22, 2023, from https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-keras