Detecting face mask by using machine learning

Yong Yu
University of Victoria
`yongyu1@uvic.ca`

Abstract

The face mask has become a daily essential for us since pandemic, we are required to wear them in any public area, but more importantly, we want to make sure other people are doing it. We want to make sure they have their mask on as well. In this paper, we addressed three different machine learning models to solve this problem, where we can detect if a person is wearing a mask or not by our models through the camera.

1 Introduction

As we addressed the problem shown in the abstract section. We will first talk about the problem and our motivation. Then, we will talk about the dataset used to train and test our models. For the dataset, we will have some preprocessing steps we need to take before we can use them in training.
We will give some introduction about the convolutional neural network and residual network. We will use both of them in the project. Then we will introduce the transfer learning technique, which allows us to adapt knowledge from the existing models.
We will describe the models we built in detail, including training time, train accuracy, test accuracy, and the models' architectures.
Finally, we will discuss the result, show the confusion matrix, and reported the best model base on the test accuracy.

2 Problems

The problem is trying to detect if a person is wearing a mask or not. In terms of machine learning, this is a binary classification problem, where the output for our model will be 0 or 1, in this project, 1 as if the person is wearing a mask, 0 otherwise.
We want to take advantage of machine learning to accomplish this problem. We will be building machine learning models and training them with our dataset.

2.1 Motivation

We want to do this project because of a few reasons. First, we will build the models that were not covered in class. We will describe them in the background section. The second reason is we want to gain more experience with computer vision problems, primarily if we can interact with, such as this project, it this project. What we need are a webcam and a face mask. The third reason will be this is such a trending topic, maybe not in a good way. This model has many real-world usages. For example, put it on the front door of the house.

2.2 DataSet

In this project, we are using the dataset provided in the following URL:

https://data-flair.training/blogs/download-face-mask-data/

This dataset contains 1376 images in total, with 690 images containing images of people wearing masks and 686 images with people without masks. After we obtain the dataset, we will have to preprocess the dataset before we use them directly in training. We have to do some preprocess steps.

- **Grayscale images:** We will load the dataset images as gray scale images, the color doesn't provide any helpful information when detecting face masks in the pictures. Colored are more complex for computers.
- **Resize the images:** By making sure our model is consistent, we make sure, all the data has the same shape. We will shape our data as (224, 224, 3). We will discuss why we resize them to this size next section when we talk about the pre-trained models we will use in this project.
- **Data augmentation:** We increase the size of the dataset by rotating them by [90, 270] degrees, we will end up with a dataset tripped the size of the original dataset. Our dataset now contains 4128 samples.
- **Labeling:** The original dataset provided all the images. However, when we doing supervised learning, we need to label the data ourselves, we will label the images with the facemask with 1 and images without face mask with 0.
- **CSV:** In order to access the images and the transformation from the code, we need to be able to store the data in some manner. We choose to store the data in a CSV file containing the following information:
  1. The path to the image.
  2. The label of the image.

Now we have the dataset that can be easily access with codes. We choose some sample images shown in Figure 1.

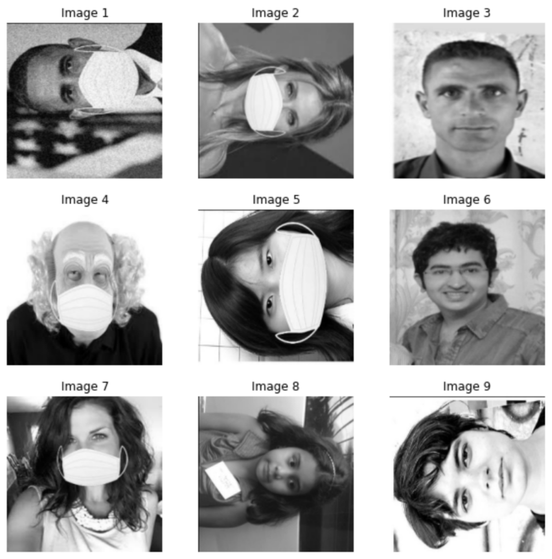


Figure 1: Samples

3 Background

To better understand the project, the model, and the concept we used. This chapter provided a background introduction to two different neural networks which are convolutional neural networks and residual neural networks. We will explain the concept of transfer learning, which allows us to adapt the knowledge from an existing model so we will need much less data to get a good result.

3.1 Convolutional neural networks(CNN)

A convolutional neural network(CNN)[1] is known to be good when dealing with image tasks. A basic convolutional neural network includes four different types of layers. They are convolutional layers, pooling layers, fully connected layers, and output layers. We show the architecture of a convolutional in Figure 2.

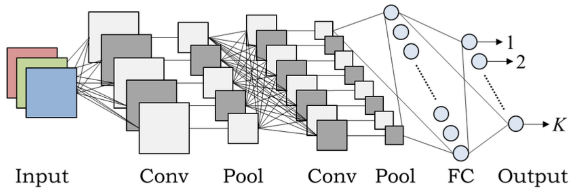


Figure 2: CNN architecture

.

We will give a brief introduction of each layer about what they do.
Convolutional layer: The convolutional layer applies different filters to the input. The earlier convolutional layers identify the simple features such as horizontal/vertical lines, corners dot... later convolutional layers identity more complex features. There are two different parameters we can set in this layer, padding, and strides. We can choose using ‘same’ padding or ‘valid’ padding. One of them shrinks the size of the input image after the filter, one of them not. The stride is an integer that tells how many pixels the filter will move each time.
Pooling layer: In general, the pooling layer will shrink the input size but try to keep its features, we can choose to do max-pooling or average-pooling. For each region represented by the filter, we take the max/average of that region and create a new, output matrix where each element is the max/average of a region in the original input. That is the different between max/average pooling.
Fully connected layer: The fully connected layer is like a layer we will see in a typical neural network. It is the output from the final pooling or convolutional layer.
Output layer: We will output the result in this layer.

3.2 Residual neural networks

The original paper of residual network came out at 2016[2]. When people are still training 20 30 layers models, the authors of this paper[2] came out with the architecture of the residual neural networks around about 50 layers and still holds good accuracy impressive.
The residual neural network is often much deeper than a convolutional neural network, but that doesn’t mean we need to train as many parameters. The residual neural network introduces the architecture call residual blocks shown in Figure 3.

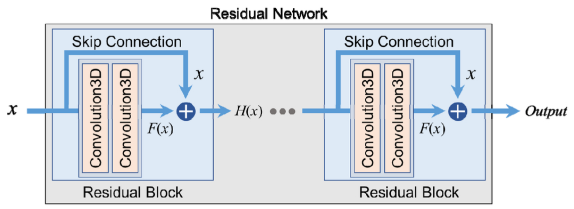


Figure 3: Residual block architecture

Unlike a traditional neural network where we pass the input layer by layer, the residual network has some skip connections.

3.3 Transfer learning

Transfer learning[3] can be very helpful with certain machine learning or data mining model. Transfer learning allows us to use the knowledge within a neural network that has already been trained and apply it to help solve a new task. That means we can find a neural network that is has been trained to do some task. We strip off the last few layers in the pre-trained neural network and add our own customized layers after that so that when we train the neural network, we only need to train the new layers that we added while retaining the knowledge of the other layers.
The problem of building and training a model from scratch is that sometimes it is too hard or is impossible to collect enough data for the model, or it takes too much resources to train and build the model from scratch. By using transfer learning, we may avoid these expenses. Transfer learning can be applied to different machine learning problems.
In transfer learning, we find a pre-trained machine learning model and we only keep the part that we need to keep in the model, and we add some new layers after. We only need to train the layers that come after the original model, which can save data and time. Figure 4 shows briefly how transfer learning works.

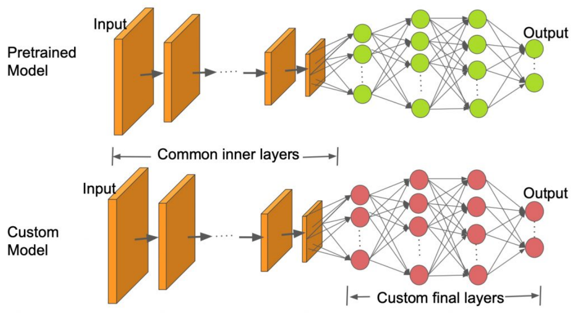
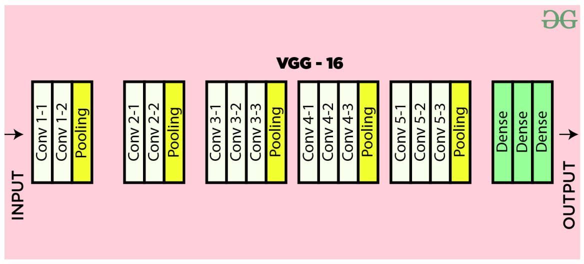


Figure 4: Transfer learning

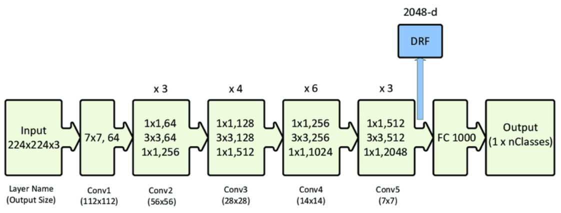
In this project, we will build two models that take advantage of transfer learning. One of them uses a pre-trained CNN, and another one uses a pre-trained ResNet(residual network). We will introduce them in the next section.

4 The models

This section will give a more detailed description of our models. There are three models we will be built. Two of them take advantage of transfer learning. We have found two pre-trained models that we will use, called VGG16[4] and ResNet50[2].We showed the architecture of the models in Figure 5.
The VGG-16[4] is a 16 layers pre-trained convolutional neural network trained with over 40 million images. It takes the input image size of (224 x 224).
The ResNet50[2] is a pre-trained residual neuron network that has total of 50 layers trained over million of images. It also takes the input image size of (224 x 224).



(a)

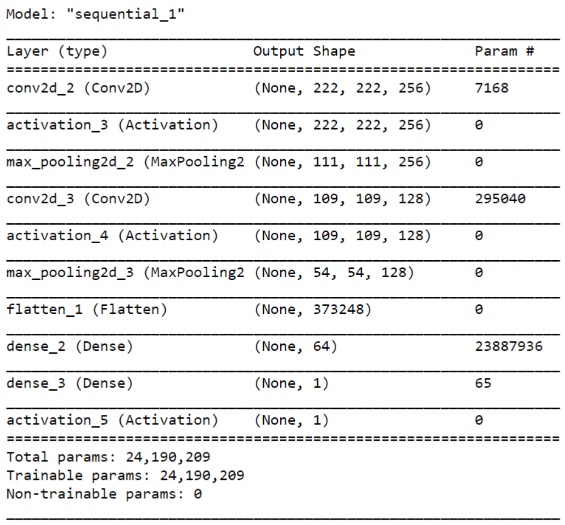


(b)

Figure 5: (a)VGG16 architecture (b)ResNet50 architecture

4.1 Base model

The first model is a plan convolutional neural network. It has two convolutional layers with two max-pooling layers, followed by the fully connected layer, one dense layer, and the output layer. The architecture of the model is shown in Figure 6. It has a total of 24,190,209 parameters to train.

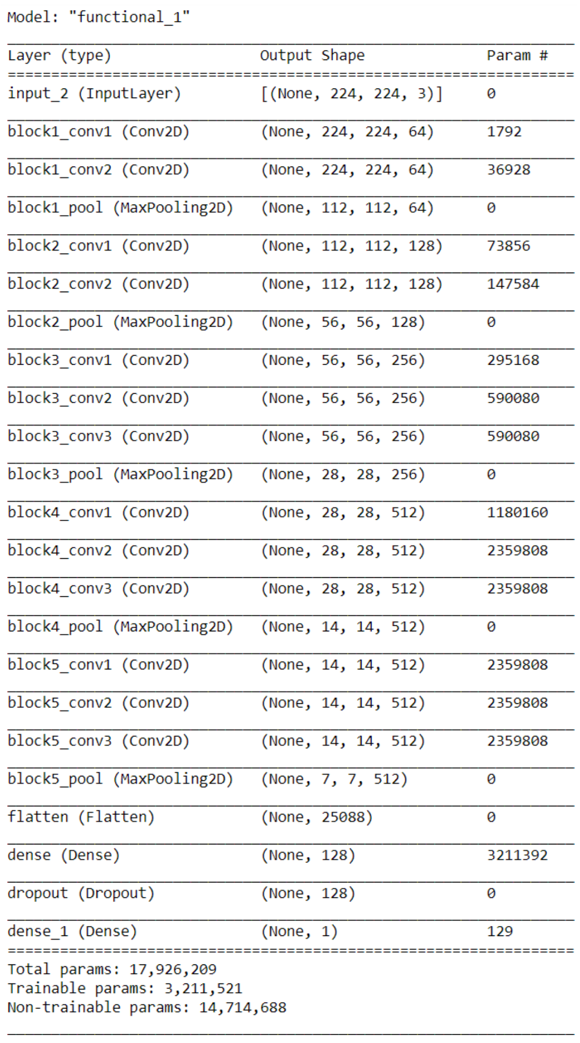


```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 222, 222, 256)     7168
_________________________________________________________________
activation_3 (Activation)    (None, 222, 222, 256)     0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 111, 111, 256)     0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 109, 109, 128)     295040
_________________________________________________________________
activation_4 (Activation)    (None, 109, 109, 128)     0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 54, 54, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 373248)            0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                23887936
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_5 (Activation)    (None, 1)                 0
=================================================================
Total params: 24,190,209
Trainable params: 24,190,209
Non-trainable params: 0
_________________________________________________________________
```

Figure 6: Base model

4.2 VGG16 base model

We build a model using transfer learning and uses the VGG16[4] as a pre-trained model. We will freeze the weight of this model. We will not include the layers after the last layer of the convolution/pooling layer in the model. We added 2 dense on top of the pre-trained model and shown the architecture of this model in Figure 7. This model has 3,211,521 parameters to trained.

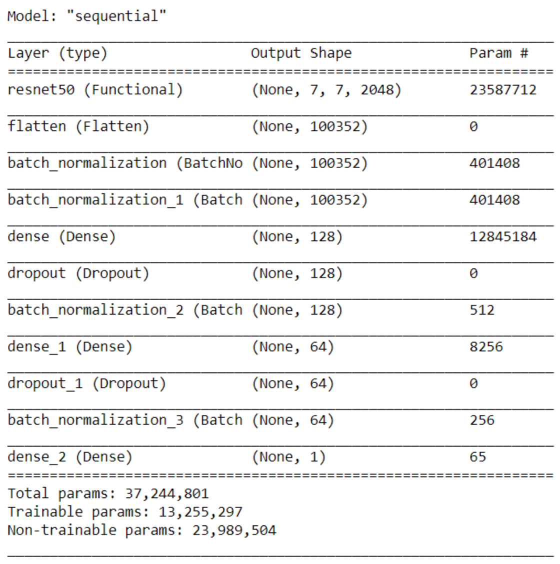


```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0         
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160   
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808   
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808   
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0         
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808   
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808   
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808   
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0         
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0         
_________________________________________________________________
dense (Dense)                (None, 128)               3211392   
_________________________________________________________________
dropout (Dropout)            (None, 128)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129       
=================================================================
Total params: 17,926,209
Trainable params: 3,211,521
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 7: VGG16 model

4.3 ResNet50 base model

Similar to the vgg16 model. This model uses ResNet50 as a pre-trained model, and it has 13,255,297 parameters shown in Figure 8.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Functional)        (None, 7, 7, 2048)        23587712
_________________________________________________________________
flatten (Flatten)            (None, 100352)            0
_________________________________________________________________
batch_normalization (BatchNo (None, 100352)            401408
_________________________________________________________________
batch_normalization_1 (Batch (None, 100352)            401408
_________________________________________________________________
dense (Dense)                (None, 128)               12845184
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
batch_normalization_3 (Batch (None, 64)                256
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 37,244,801
Trainable params: 13,255,297
Non-trainable params: 23,989,504
_________________________________________________________________
```

Figure 8: ResNet50 model

5 Experiment and result

This section describes how we set up this experiment and the experiment result.

5.1 Training

After we preprocess our dataset. Our training dataset contains a total of 3939 samples. The test dataset sontains 579 samples. We will use 10% of the data from training set as validation data. The new dataset size as describe as follow:

- **Train data:** 3545 samples.
- **Validation data:** 394 samples.
- **Test data:** 579 samples.

Now we have the dataset, we will report the total training time that takes to train each model on an **i5 CPU**.

- **Base model:** 3h 19min 14s.
- **VGG16 base model:** 2h 1min 10s.
- **ResNet50 base model:** 1h 42s.

5.2 Result

In this section, we will report the train and test accuracy for each model, and base on the accuracy, we pick the model with the best accuracy as the best model for our project.

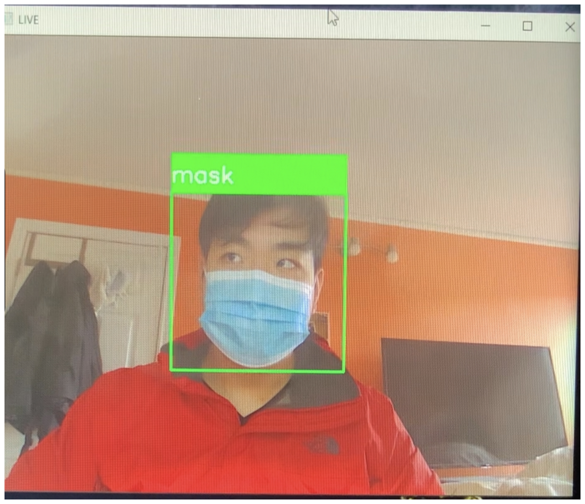
- **Base model.**
  - Train accuracy: **0.4996**
  - Test accuracy:**0.4974**
- **VGG16 base model.**
  - Train accuracy:**0.9939**
  - Test accuracy:**0.9862**
- **ResNet50 base model.**

- Train accuracy:**0.8191**
- Test accuracy:**0.8497**

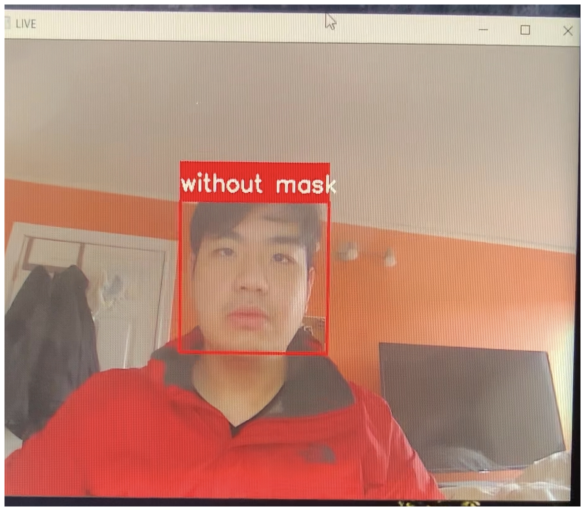
Base on the train and test accuracy, the **VGG16 base model** has the highest accuracy for both train and test dataset. The **VGG16 base model** will be the best model to pick for this project.

6 Conclusion

We developed three different machine learning models to detect if a person is wearing a mask or not. We preprocessed the dataset for better training experiments.
We introduced two different neural network types, convolutional neural networks[1] and residual neural networks[2] . We also introduced the concept of transfer learning[3] . Two of the models we built uses transfer learning.
We introduced two pre-trained models that we use during our implementation. VGG16[4] and ResNet50[2].
We trained the models and picked the model for our problem, which comes out to be the VGG16 base model. We tested our model in real world and the result shown in Figure 9.



(a)



(b)

Figure 9: (a)With mask (b)Without mask

As we can see the model classify the images successfully.

6.1 Future work

During the training time, I was using the cpu, so it takes a while to train, we can lower the training time by GPU or an online server such as AWS.
We can try to integrate the model into software as long we can access with proper API.
The accuracy can be future improve by collecting more data or adding more layers to our models. We will need more resource to train them.

References

[1] ALBAWI, S., MOHAMMED, T. A., AND AL-ZAWI, S. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)* (2017), Ieee, pp. 1–6.

[2] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[3] PAN, S. J., AND YANG, Q. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering 22*, 10 (2009), 1345–1359.

[4] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).