Face Mask Detection

蔡承哲(Cheng-Zhe Cai)*
Department of Industrial Management
National Taiwan University of Science and Technology Taipei,Taiwan
M11001105

黃名瑋(Ming-Wei Huang)*
Department of Industrial Management
National Taiwan University of Science and Technology Taipei,Taiwan
M11001209

***Abstract*—**

Keywords—CNN, InceptionRestNetV2, VGG16, Face Mask classification

I. INTRODUCTION

Currently, the world is suffering from a pandemic of a virus called COVID-19. COVID-19 is an infectious disease caused by severe acute respiratory syndrome (SARS-CoV-2). People can get the virus if they have close physical contact with an infected person through respiratory droplets while coughing, sneezing and/or talking. The virus can be spread by touching a surface or object with the virus, and then we touch our mouth, nose or eyes with our hands. Now, we can protect ourselves by not getting close to the virus. According to the CDC, the best ways to avoid the spread of the virus are social distancing and the use of masks in public places.

Therefore, mask testing is now essential in public spaces. With the help of current technology, with the help of deep learning, it will be easy to detect the use of masks. Research related to mask testing is no longer restricted. In our case, we used two CNN architectures that VGG16 and InceptionRestNetV2 to identify whether people are wearing masks and wearing them correctly.

II. RELATED WORK

In the past few years, large progress has been made on FAC. Traditional FAC methods, rely on handcrafted features to perform attribute classification. With the development of deep learning, current state-of-the-art FAC methods employ CNN models to predict the attributes and have shown remarkable improvements in performance. Our proposed method is closely related to CNN-based learning, multi-label learning and attribute grouping. In this section, we briefly introduce related work based on CNN.

A. Face Mask Detection

A. Chavda, J. Dsouza, S. Badgujar, A. Damani (2020) in these paper authors have proposed a two-stage architecture. Stage 1 is a face detector that acts as the first stage of the system. A raw RGB image is transferred as input to this stage. The face detector extracts and generates all recognized faces in the image with their bounding box coordinates. Accurate facial recognition is very important to our architecture. Training a high-precision face detector requires a great deal of tagged data, time, and computational resources. Level 2 The second level of the system is a mask sorter. In this phase, the processed ROI is taken from the intermediate processing block and classified in such a way that the dataset also contains images of misused face masks or hands covering the face, which are classified as unmasked faces [1].

B. Multi-task Learning

Multi-task Learning (MTL) is an effective learning example to improve the performance of a main task with the help of some related additional tasks. The CNN model can be easily used for MTL, and the tasks learn general feature representations in the deep layers. MTL has proven to be effective in different computer vision works. For example, Tan et al. learn multiple attention mechanisms in an MTL manner for pedestrian attribute analysis. Zhang et al. perform FLD together with some related tasks, such as gender classification and gesture recognition. It seems that assigning parameters' weights to different loss functions for multi-task deep learning is important for building model. Kendall et al. propose to weigh loss functions based on the homoscedastic uncertainty of each task, where the weights are automatically learned from the data. Recently, Liu et al. develops a multi-task attention network. Automatically learns both task-shared and task-specific features in an end-to-end manner, for MTL. They develop a novel weighting scheme, Dynamic Weight Average (DWA), which learns the weights based on the rate of loss changes for each task.

C. Multi-label Learning

Traditional CNN based FAC methods mainly use single-label learning to predict facial attributes. For example, Liu et al. propose to cascade two Localization Networks (LNets) and an Attribute Network (ANet) to localize face regions and extract features, respectively. They use the features extracted from ANet to train 40 SVMs to classify 40 attributes. The single-label learning based FAC methods consider the classification of each attribute as a single and independent problem, thereby ignoring the correlations among attributes. Moreover, these methods are usually time consuming and cost prohibitive.

Multi-label learning based FAC methods predict multiple facial attributes simultaneously in an end-to-end trained network. Because each face image is naturally associated with multiple attribute labels, multi-label learning is actually suited for FAC. For example, Ehrlich et al. use a Restricted Boltzmann Machine (RBM) based model for attribute classification. Rudd et al. introduce a Mixed Objective Optimization Network (MOON) to address the multi-label imbalance problem. Huang et al. propose a greedy neural architecture search method to automatically discover the optimal tree-like network architecture, which can jointly predict multiple attributes.

Nowadays, multi-label learning based FAC methods, which apply the same network structure for each attribute, usually learn the features of facial attributes on the upper layers of CNN. However, various facial attributes have different learning models. Hence, it is more worth of developing a new CNN model. We should consider the diverse learning complexities of attributes rather than using same pattern to the attributes during the training datasets.

III. Approach

A. Image

The image we use are from the dataset named Face Mask Detection on the kaggle website .The original dataset contains three classes : with mask, without a mask, and wearing mask incorrectly, in which the number of image in each class is 2994.

In our study, we only use 2000 images of each class in the dataset ,which are the images numbered 1~2000 in each class of images.

B. Training and test sets

Before we train the model, we have to resize the pictures first. In order to make our training process more quickly, we decide to resize our picture into resolution 128*128.

C. Face Mask Detection

- RestNet and Inception Module

Residual learning is easier than direct learning from raw features and it can avoid the degradation problem. When the residual is 0, the accumulation layer only does the identity mapping at this time, at least the network performance will not be degraded, in fact the residual will not be 0 which will also enable the accumulation layer to learn new features based on the input features, resulting in better performance.[2]

The basic structure of the Inception Module has four components. 1×1 convolution, 3×3 convolution, 5×5 convolution, 3×3 max pooling. Finally, the results of the four component operations are combined on the channel. This is the core idea of Inception Module. The information of different scales of the image is extracted through multiple convolution kernels, and finally fused, a better representation of the image can be obtained.

- InceptionRestNetV2

InceptionResNetV2 is the model we use. It combines with Inception net and Residual net. Fig.1. shows the InceptionResNetV2 architecture.

Model: "inception_resnet_v2"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 512, 512, 3)	0
conv2d_1 (Conv2D)	(None, 255, 255, 32)	864
batch_normalization_1 (BatchNor	(None, 255, 255, 32)	96
activation_1 (Activation)	(None, 255, 255, 32)	0
conv2d_2 (Conv2D)	(None, 253, 253, 32)	9216
batch_normalization_2 (BatchNor	(None, 253, 253, 32)	96
activation_2 (Activation)	(None, 253, 253, 32)	0
conv2d_3 (Conv2D)	(None, 253, 253, 64)	18432
batch_normalization_3 (BatchNor	(None, 253, 253, 64)	192
activation_3 (Activation)	(None, 253, 253, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 126, 126, 64)	0
conv2d_4 (Conv2D)	(None, 126, 126, 80)	5120
batch_normalization_4 (BatchNor	(None, 126, 126, 80)	240
activation_4 (Activation)	(None, 126, 126, 80)	0
conv2d_5 (Conv2D)	(None, 124, 124, 192	138240
batch_normalization_5 (BatchNor	(None, 124, 124, 192	576
activation_5 (Activation)	(None, 124, 124, 192	0
max_pooling2d_2 (MaxPooling2D)	(None, 61, 61, 192)	0
conv2d_9 (Conv2D)	(None, 61, 61, 64)	12288
batch_normalization_9 (BatchNor	(None, 61, 61, 64)	192
activation_9 (Activation)	(None, 61, 61, 64)	0
conv2d_7 (Conv2D)	(None, 61, 61, 48)	9216
conv2d_10 (Conv2D)	(None, 61, 61, 96)	55296
batch_normalization_7 (BatchNor	(None, 61, 61, 48)	144
batch_normalization_10 (BatchNo	(None, 61, 61, 96)	288
activation_7 (Activation)	(None, 61, 61, 48)	0
activation_10 (Activation)	(None, 61, 61, 96)	0
average_pooling2d_1 (AveragePoo	(None, 61, 61, 192)	0

Fig.1.An illustration of the architecture of InceptionResNetV2

- VGG16

The VGG network architecture was initially proposed by Simonyan and Zisserman [3]. Figure.2. shows the six structures in the VGG paper [3], which are distinguished by different convolution kernel sizes and neural network layers. The VGG models with 16 layers (VGG16) and with 19 layers (VGG19) were the basis of their ImageNet Challenge 2014 submission.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Fig.2. An illustration of the architecture of VGG

The VGG16 architecture, shown at the top of Fig.2, is structured starting with five blocks of convolutional layers followed by three fully-connected layers. In VGG16, the input to cov1 layer is of fixed size 224x224 RGB image. Convolutional layers use 3×3 kernels with a stride of 1 and padding of 1 to ensure that each activation map retains the same spatial dimensions as the previous layer. The advantage of using smaller convolution kernels is that by using smaller convolution kernels (ex: 3x3) instead of larger convolution kernels (ex: 5x5, 7x7), the network can be improved with the same receptive field depth.

A rectified linear unit (ReLU) activation is performed right after each convolution and a max pooling operation is used at the end of each block to reduce the spatial dimension. Max pooling layers use 2×2 kernels with a stride of 2 and no padding to ensure that each spatial dimension of the activation map from the previous layer is halved. Two fully-connected layers with 4096 ReLU activated units are then used before the final 1000 fully-connected softmax layer.

In summary, because the architecture of VGG16 has the characteristics of simplicity and unity, we decided to use VGG16 for our study.

- Transfer learning

Transfer learning consists in transferring the parameters of a neural network trained with one dataset and task to another problem with a different dataset and task [4].

In this study, we adopt a transfer learning approach since the previous studies have shown in order to avoid either overfitting or underfitting consequences using a small training dataset, a better approach is to take advantage of a CNN initially trained using a large-scale dataset [5].

In this study, we select a VGG16 model, which was pre-trained on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) using a large dataset with 14 million images [6].

IV. Experiments

A. Methodology

- Platform used

We use google colab to run this code. Google colab is a free CPU virtual machine based on Jupyter Notebook. You can write programs through a browser and connect to Google Drive to store the trained AI model and use GPU or TPU for free.

- Training and testing splits

In this study, we used 3000 images as the training dataset, including 1000 images of each class, and 1500 images as the validation dataset, including 500 images of each class. Finally, we use 1500 images as the test dataset.

- Image generator

Regarding to the scarcity of the dataset we filmed by ourselves, compared to other mature models' training dataset, we use image generator to construct familiar pictures to add additional images to our dataset while prevent overfitting issues that may occur in our training model.

```
train_datagen = ImageDataGenerator(rescale=1./255,rotation_range=20,
                          width_shift_range=0.2,
                          height_shift_range=0.2,
                          shear_range=0.2,
                          zoom_range=0.5,
                          fill_mode="nearest")
```

- InceptionRestNetV2

The InceptionrestNetV2 model we had adjusted three parameters: Loss function, Optimizer and metric.

```
x = model.output
x = GlobalAveragePooling2D()(x)
predictions = Dense(3, activation='softmax')(x)
model = Model(inputs=model.input, outputs=predictions)

model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

To train the model to the best of its ability, we set the initial learning rate to 0.0001 and if there is no improvement after 5 epochs of training, the learning rate will be reduced by multiplying by 0.5. When the loss does not drop within 10 epochs, the model will stop training to avoid overfitting. In addition, we also set model storage conditions to save the best model.

```
checkpoint = ModelCheckpoint('InceptionResNetV2_checkpoint_v2.h5', verbose=1,
                         monitor='val_loss', save_best_only=True,
                         mode='min')
reduce_learning_rate= ReduceLROnPlateau(monitor='val_loss', factor=0.5,
                          patience=5, mode='min', verbose=1,
                          min_learning_rate=1e-6)

estop = EarlyStopping(monitor='val_loss', patience=10, mode='min', verbose=1)
```

- VGG16

Because we are using the transfer learning method, we first load the pre-trained VGG16 model, then remove the top 3 fully-connected layers, and only use VGG16 to extract features. We freeze those layers of VGG16 because we only need to train the last layer, and finally, add the last layer and set our output to only have 3 classes.

```
model = Sequential()
model.add(VGG16(weights='imagenet',pooling='avg',include_top=False,)
model.add(Dense(num_classs, activation='softmax'))
model.layers[0].trainable = False
```

Our model has three parameters: Loss function, Optimizer, metric.

First, the Loss function is to help judge the error value because we are doing multivariate classification, so our loss function uses categorical crossentropy.

Second, the function of the Optimizer is to help the neural network to adjust the parameters. The purpose is to make the loss as small as possible. Common Optimizers include Momentum, AdaGrad, and Adam. Momentum means "movement". This optimizer is a simulation The concept of physical momentum, the learning speed will be faster in the same direction dimension, and the learning speed will be slower when the direction changes, and Adagrad restricts the learning rate according to the customized value of each parameter, and adjusts the learning according to the gradient Rate. The advantage is that the training speed can be accelerated, the gradient can be enlarged when the gradient is small in the early stage, and the gradient can be constrained when the gradient is large in the later stage, but the disadvantage is that the gradient may approach 0 in the middle and late stages of training, and end prematurely, while Adam Optimizer It is a combination of Momentum and AdaGrad. Adam retains Momentum to adjust the gradient speed of the past gradient direction and Adam to adjust the learning rate of the square value of the past gradient, plus Adam has the parameter "deviation" Correction", so that each learning rate will have a certain range, which will make the parameter update more stable.

In summary, we use Adam as our Optimizer.

Third, metric is used to evaluate the performance of the current training model, we use 'accuracy' to evaluate the accuracy of the prediction

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

B. Parameters

- Epoch.

In terms of artificial neural networks, an epoch refers to one cycle through the full training dataset. Usually, training a neural network takes more than a few epochs. In other words, if we feed a neural network the training data for more than one epoch in different patterns, we hope for a better generalization when given a new "unseen" input (test data). An epoch is often mixed up with an iteration. Iterations is the number of batches or steps through partitioned packets of the training data, needed to complete one epoch.it gives the network a chance to see the previous data to readjust the model parameters so that the model is not biased towards the last few data points during training.

Determining how many epochs a model should run to train is based on many parameters related to both the data itself and the goal of the model.

In brief, we choose epoch = 20 to improve accuracy.

- Initial Learning rate.

The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated.

In setting a learning rate, there is a trade-off between the rate of convergence and overshooting. While the descent direction is usually determined from the gradient of the loss function, the learning rate determines how big a step is taken in that direction. A too high learning rate will make the learning jump over minima but a too low learning rate will either take too long to converge or get stuck in an undesirable local minimum.

In order to achieve faster convergence, prevent oscillations and getting stuck in undesirable local minima the learning rate is often varied during training either in accordance to a learning rate schedule or by using an adaptive learning rate.

The issue with learning rate schedules is that they all depend on hyperparameters that must be manually chosen for each given learning session and may vary greatly depending on the problem at hand or the model used. To combat this there are many different types of adaptive gradient descent algorithms such as Adagrad, Adadelta, RMSprop and Adam which are generally built into deep learning libraries such as Keras.

In brief, we choose initial learning rate = 1e-5 to improve accuracy.

- Batch Size.

The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters. In other words, that batch size refers to the sample size used in 1 iteration, and the training dataset can be divided into one or more batches.

We choose batch size = 20 to improve accuracy.

C. *Results and Evaluation*

In this experiment, we have a total of 1500 test data, of which 507 are pictures of incorrectly wearing masks, 497 are pictures with masks, and 496 are pictures without masks. we used the Confusion Matrix to observe the results of the experiment.

- InceptionResNetV2

The confusion matrix of InceptionResNetV2 model(Fig.3.) of the 507 pictures of incorrectly wearing masks, the system judged that 501were incorrect, 490 of the 497 pictures with masks are predicted to mask, and 470 of the 496 pictures without masks are predicted to without mask. In the test set, the accuracy of InceptionResNetV2 is 0.9967 and the loss is 0.0093.

Initially, we set the epochs to 20 times. We can see from the Fig.4., finally the model was training completed for 15 epochs and stopped training.

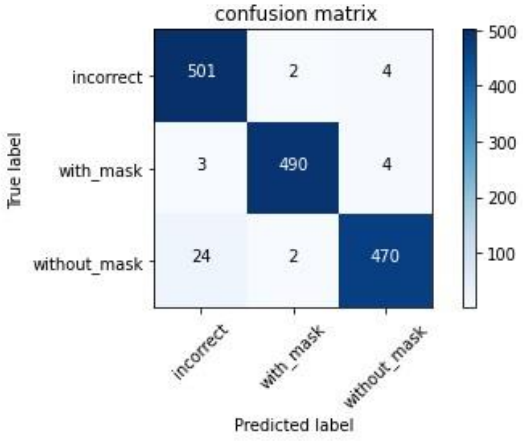

Fig.3. The confusion matrix of the InceptionResNetV2 model.

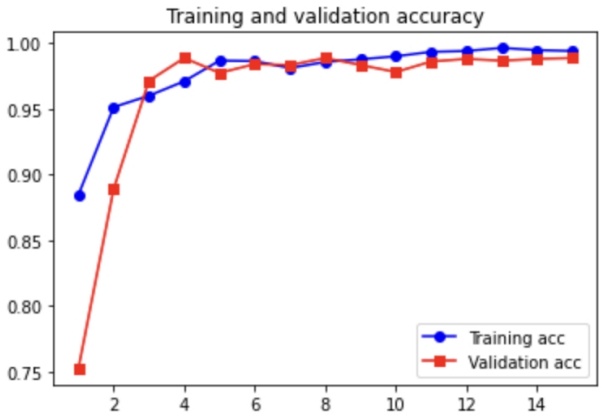

Fig.4. InceptionResNetV2 model accuracy of training and validation.

- VGG16

In this confusion matrix of VGG16 model(Fig.5.) of the 507 pictures of incorrectly wearing masks, the system judged that 500 were incorrect, 469 of the 497 pictures with masks are predicted to mask, and 485 of the 496 pictures without masks are predicted to without mask.

All correct predictions are located in the diagonal of the table, so it is easy to visually inspect the table for prediction errors, as they will be represented by values outside the diagonal.

According to Fig.5., in the test set, the accuracy of VGG16 is 0.9693 and the loss is 0.2891.

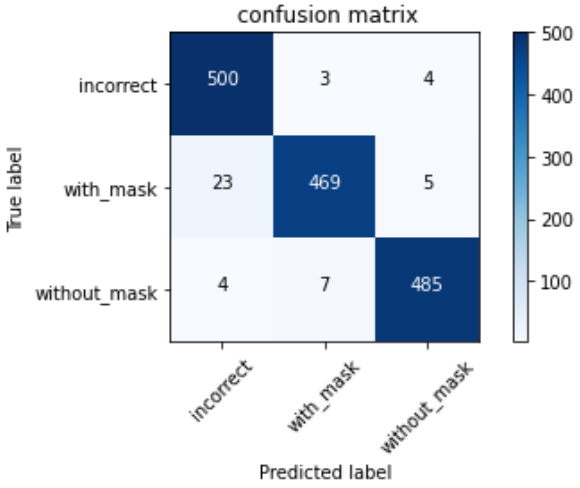

Fig.5. The confusion matrix of the VGG16 model.

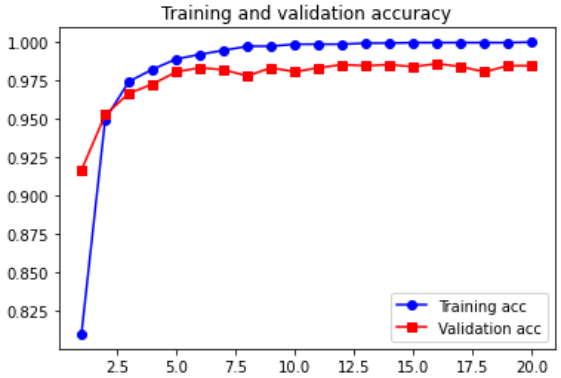

Fig.6. VGG16 model accuracy of training and validation.

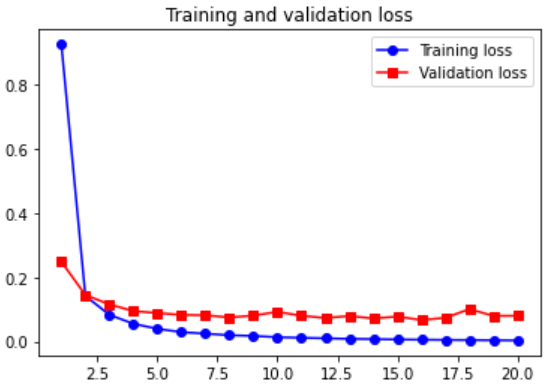

Fig.7. VGG16 model loss of training and validation.

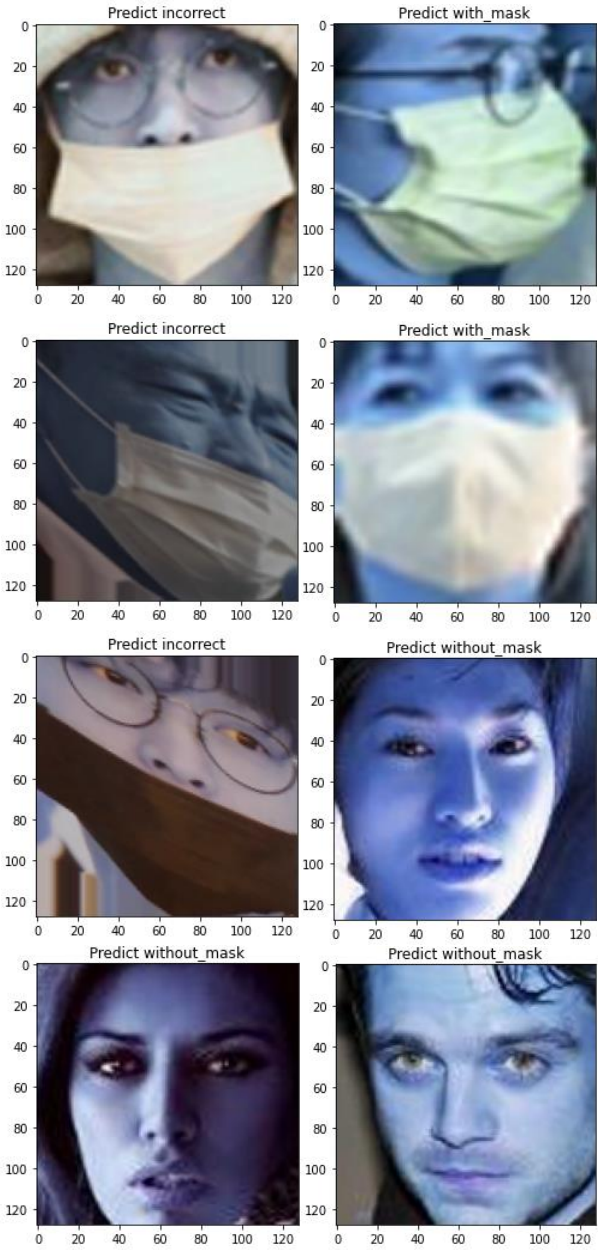

Fig.8. Predicted results of VGG16 model

V. CONCLUSION

In this mask detection dataset experiment, we need to detect the condition of a person wearing a mask, including whether the mask is worn or whether the mask is worn correctly. In the pre-processing stage of this experiment, because the amount of data is small, we use image enhancement methods, including rotation, zoom, etc., to increase the number of our training data. We used two models, VGG16 and InceptionResNetV2, for our mask detection. The results show that both models perform well, with an accuracy of over 95%. Among them, the accuracy of InceptionResNetV2 is nearly 100%, so it is a very suitable model for mask detection.

REFERENCES

[1] Gagandeep Kaur, Ritesh Sinha, Puneet Kumar Tiwari, Srijan Kumar Yadav, Prabhash Pandey, Rohit Raj, Anshu Vashisth, Manik Rakhra, Face mask recognition system using CNN model,Neuroscience Informatics,Volume 2, Issue 3,2022.

[2] Sibo Qiao, Shanchen Pang, Gang Luo, Silin Pan, Zengchen Yu, Taotao Chen, Zhihan Lv,RLDS: An explainable residual learning diagnosis system for fetal congenital heart disease,Future Generation Computer Systems,Volume 128,2022,Pages 205-218.

[3] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[4] J. Yosinski, J. Clune, Y. Bengio, H. Lipson, How transferable are features in deep neural networks? in Advances in Neural Information Processing Systems (2014), pp. 3320–3328.

[5] Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, *22*(10), 1345-1359.

[6] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*(3), 211-252.

[7] Heidari, M., Mirniaharikandehei, S., Khuzani, A. Z., Danala, G., Qiu, Y., & Zheng, B. (2020). Improving the performance of CNN to predict the likelihood of COVID-19 using chest X-ray images with preprocessing algorithms. *International journal of medical informatics*, *144*, 104284.

[8] Rezende, E., Ruppert, G., Carvalho, T., Theophilo, A., Ramos, F., & de Geus, P. (2018). Malicious software classification using VGG16 deep neural network's bottleneck features. In *Information Technology-New Generations* (pp. 51-59). Springer, Cham.