COMP 6721 project assignment

Part1:

https://github.com/Xzxleah/COMP6721/blob/main/Code/comp6721_project_part1.ipynb

Part2:

https://github.com/Xzxleah/COMP6721/blob/main/Code/comp6721_project_part2_ipynb.ipynb

Muyi Chen(40202802) data processing

Junteng Ma(40204015) training

Zhixian Xiang(40017081) evaluation

bias k-fold

1 Dataset and data processing

In this project, we are going to accomplish an image classification task by using deep learning models. Fetching the dataset is critical for a deep learning project. The image data we used in the project is found and can be directly downloaded from the kaggle dataset and then import into the Colab environment. Among all the images, we have selected 2000 mask-related images which are separated into 5 categories, including 'cloth mask', 'N95 mask', 'no mask', 'surgical mask' and 'mask worn incorrectly'. Images from first four categories are collected through real-time data [4][3][1][5][6][2] and images from the last category('mask worn incorrectly') are collected through generated data[7]. Most images from our dataset are fetched from [6]. [6] provides single face images with masks which could be separated into 'cloth mask', 'N95 mask', 'surgical mask'. Also, it provides single face images from 'no mask' category. [4] provides data from two categories 'with mask' and 'without mask' and we are able to manually select classes 'cloth mask', 'N95 mask' and 'surgical mask' from the original 'with mask' category. [3] provides images in three categories. However, the dataset provides some images with multiple people wearing masks and upon this situation, images are cropped to ensure that there is only one people in one image. For those images which include whole body rather than merely the face, we also apply cropping to remain only face in the image. MaskedFace-Net[7] provides generated 'mask wear incorrectly' images. Some Samples from each categories are shown in Fig.1. All images are finally transformed into .png format for further data processing.

Figure 1: image samples from each category

The dataset is stored under the same folder with the source code. Under the root folder naming 'mask_dataset', there are five folders mapping to five classes. The structure of the dataset is shown in Fig.2. Each class has 400 images and there 2000 images in the dataset in total. As a result, there are 400 images in each folder.

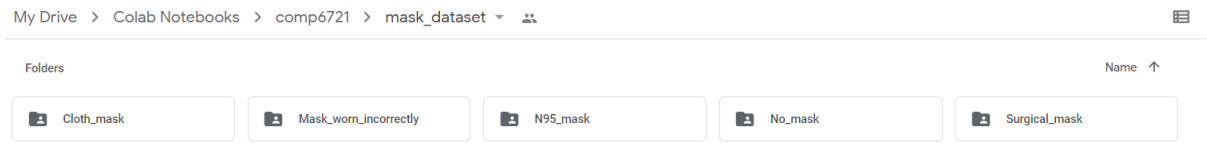

Figure 2: structure of the dataset

During data preprocessing, 1500 images are split into the training dataset and others are split into the test dataset. We apply 'random_split' provided by pytorch to do train-test splitting. After the splitting, all datasets are almost equal sized. However, there are

still some differences between size of each classes which are relatively small comparing to the total size of training and testing dataset.

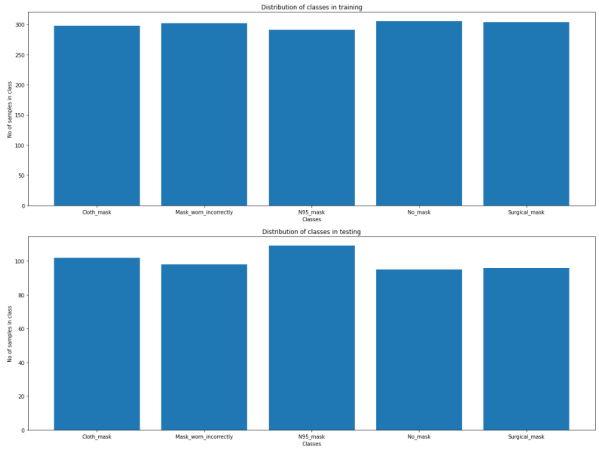

Figure 3: Number of data in training and testing dataset from each categories

On the data processing part, images are processed by transformation provided by torchvision[1]. Images are firstly normalized through transforms.Normalize and then are shaped into size (128×128) by transforms.Resize to be equal sized and then they could be treated as input into Deep learning models. Also, the datasets are loaded with ImageFolder provided by torchvision.

[1]https://pytorch.org/vision/stable/transforms.html

2 Training

Table.1 gives an overview of CNN structures used in this project. As shown in table, we use three deep learning models in this project, naming CNN, CNN with less pooling and deeper CNN. In the table, conv$a-b$ denotes a 2-dimensional layer with kernel size $a \times a$ and output channels b and maxpool$a$ denotes a max pooling layer with kernel size $a \times a$ and stride a. FC $-$ a are fully connected layers with output dimension a. For every models, convolutional layers are firstly used to extract features. Between convolutional layers, pooling layers are used to reduce the output size of output features. After features are extracted, they will be treated as input of later fully connected layers. The fully connected layers will process the features by increasing their dimensions and then a vector with the same size of number of classes is produced. Finally, softmax function is used to produce possibilities.

Table 1: CNN structures

<table>
<tr><td colspan="3">input (128x128 RGB image)</td></tr>
<tr><td>CNN</td><td>CNN with less pooling</td><td>deeper CNN</td></tr>
<tr><td>conv3-64</td><td>conv3-64</td><td>conv3-64</td></tr>
<tr><td>maxpool2</td><td rowspan="5">conv5-128
(stride 2)</td><td>maxpool2</td></tr>
<tr><td rowspan="4">conv5-128</td><td>conv5-128</td></tr>
<tr><td>maxpool2</td></tr>
<tr><td>conv3-256</td></tr>
<tr><td>conv3-256</td></tr>
<tr><td colspan="3">AdaptiveAvgPool 7x7</td></tr>
<tr><td colspan="3">FC-250</td></tr>
<tr><td colspan="3">FC-5</td></tr>
<tr><td colspan="3">softmax</td></tr>
</table>

In each 2-d convolutional block, we add batch norm operation and relu activation function to avoid gradient vanishing and introduce nonlinearity. The batch norm operation apply normalization among each batch to ensure that the numbers are in similar numerical scale. Thus, it can prevent a number being too layer resulting to its weight being too small and the gradient of its weight decreasing to nearly zero. If the gradient stays low, there will not be many updates on the neural network during each training epoch. In the normal CNN model, there are four weighted layers, including two convolutional layers and two fully connected layers. The first convolutional layer includes a 3 convolutional kernel with stride 1. It is able to extract features in a small local range. The second convolutional layer includes a 5×5 convolutional kernel instead of the 3×3 kernel in order to extract features in a wider range. Two fully connected layers are used to project the number of features into the number of classes. As there is a huge gap between the number of features and the number of classes, a hidden layer is added rather than directly projection. There are two pooling layers in the CNN. The first max pooling layer is used to reduce size of output and the second adaptive average pooling layer not only reduce size but also ensure the output feature of each channels have fixed size of 7×7

introducing convenience for further fully connected layers. In CNN with less pooling, the first max pooling is removed and the stride of second convolutional block is increased to 2 which reduces the number of operations as we have limited GPU memory. The deeper CNN extends CNN blocks after the second CNN blocks in the normal CNN model. It firstly reuse the structure of normal CNN before the adaptive pooling layer and add two 3×3 layers. Between adding these layers, a max pooling is added which is similar to previous cases. Then, two convolutional layers with higher output channels are added to extract more features. Overall, the deeper CNN model has 6 weighted layers including 4 convolutional layers and 2 fully-connected layers.

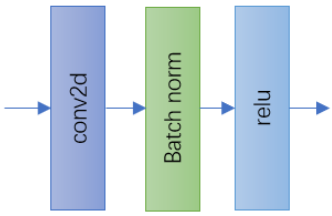

Figure 4: structure inside a CNN block

During the training process, Adam optimizer is used with the learning rate of 0.0001. Adam is a popular optimizer these days replacing tradition SGD optimizer. It combines momentum and root mean square prop and it has good performance in most cases. In our case, the learning rate is set to a relatively small number to help the model achieve local minima. If the learning rate is set too high, it is likely that the model jump over several local minima in the parameter space in each epoch and this will lead the model hard to convergence. The batch size is set to 500 which is the maximum value that we can set based on our limited resources. Meanwhile, the number of epoch is set to 5 which is original set to 10. However, it is set too high, it seems like the model will overfit the training dataset and there will be large gap between training accuracy and testing accuracy. As a result, we try different epochs decreasing from 10 and we finally end with 5 which is considered as a proper training epoch value in our case.

3 Evaluation

Table .2 provides us with a comparison of the overall performance of the three CNN models we used in the project. We evaluate the performance using four performance metrics which are accuracy, f1 score, precision, and recall. These performance metrics help us to understand the strengths and limitations of these three CNN models when we make predictions on new datasets.

Table 2: performance of each model on testing dataset

	accuracy	f1 score	precision	recall
normal CNN	77.4%	0.776	0.782	0.778
CNN with less pooling	77.6%	0.777	0.777	0.780
deeper CNN	81.0%	0.814	0.814	0.815

As shown in the table above, the deeper CNN model has the best performance among all models, with the highest values for all four-performance metrics, while the normal CNN model is the second and the model with less pooling has the lowest performance metrics values. According to the design of the network structures in the three CNN models, we can infer that with fewer pooling layers, the overall accuracy, f1 score, precision, and recall values is pretty similar compared to the normal one , while with the more extended weighted layers, the overall performance of the model has been improved and the overall value is increased by about 4%.

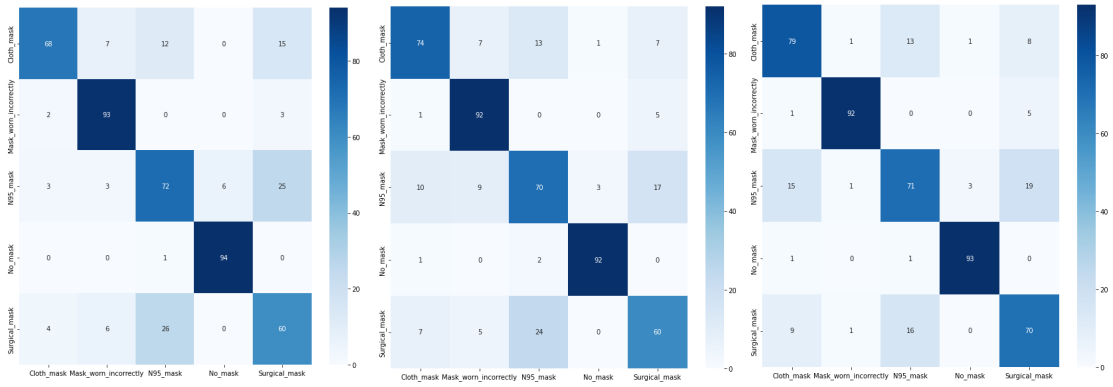

Figure 5: confusion matrix for each model
(left: normal CNN, center: CNN with less pooling layers, right: deeper CNN)

Overall, deeper CNN has a better performance with higher accuracy and f1 score. It seems like removing pooling layers has a higher impact on the performance. However, during our training process, we notice that there are fluctuations in the performances of models and the impact of removing pooling layers is not as significant as shown in this case. Also, as the training data is balanced, the f1 score has a similar value with precision and recall in all three models. The predicted mask data results of the three CNN models are shown in the above confusion matrix accordingly which uses the accuracy value to classify the test data into its corresponding labels. According to Fig.5, it is noticeable that all three models are able to distinguish class 'Mask_worn_incorrectly' and 'no_mask'

from other classes. A potential reason is that we are using generated images in the class 'Mask_worn_incorrectly' and 'no_mask' does not include masks. However, all three models cannot perform well in classifying the class 'N95_mask' and class 'surgical_mask' which are also hard to be distinguished by humans.

Performance is an important indicator for evaluating whether a CNN-based classification method is excellent, so it is important for us to know which factors affect performance. In particular, image classification performance is not only affected by the network structure itself, but also by the size of our training dataset.

CNN models normally learn features automatically from the data, and the performance is generally better when there are a lot of training data. Therefore, to improve our model performance, we can consider increasing the size of our datasets. For example, we could use Image augmentation parameters which can help us to generate more data samples using zoom, shear, rotation, preprocessing function, etc. Until now, all our three CNN models simply automate splitting the entire datasets into a training and a validation set by percentages. However, we can use K-fold cross-Validation instead, which would give us a less biased model. In addition, in order to make our model have better performance, we can also consider modifying the tuning parameters in the training network such as the number of epochs, learning rate, etc. We can also use dropout layers at each update of the training phrase in the CNN models. Or according to the needs of the model, a suitable optimizer maybe used during the model compilation process to adjust the model parameters. For example, the SGD, rmsprop, etc.

In the second part of the project, we are trying to improve the performance of deeper CNN trained before.

4 Bias

AI bias is a common situation in machine learning due to the prejudiced assumptions made during the model development processes or even biases in the training data. Our AI model is built on the people's face images; therefore, biases may also happen in our models, technically, which may arise from the following three aspects: age, race, and gender. In order to detect the presence of bias and to improve and prevent it, we decided to reanalyze our model from age and gender categories. The Age category will be divided into 3 classes: child, adult, and old while the gender category will be divided into female and male classes. Each of the divided classes includes the original five classes (cloth mask, surgical mask, FFP2/(K)N95 mask, no mask, wrong mask).

To test the age bias of the CNN model, we test with 500 adult images (100 images in 5 types of mask classes respectively), 103 child images (26 images for cloth mask, 46 images for incorrectly worn mask, 10 images for n95 mask, 1 for no mask and 20 images for surgical mask) and 123 old people images (26 images for cloth mask, 33 images for incorrectly worn mask, 18 images for n95 mask, 21 for no mask and 25 images for surgical mask) respectively. As shown in Table.3, the model accuracy of the child images and the old people images drops to around 70%, while the model accuracy for adult images is 84.6%. One reason for the decrease of the model acccuracy of the child and the adult could be the lack of sufficient images of children and old people groups. This also implies that different age groups may have some effects on the model bias.

Table 3: performance of model on different age classes

	accuracy	f1 score	precision	recall
children	70.0%	0.699	0.715	0.700
adult	84.6%	0.845	0.850	0.846
old	82.0%	0.821	0.824	0.820

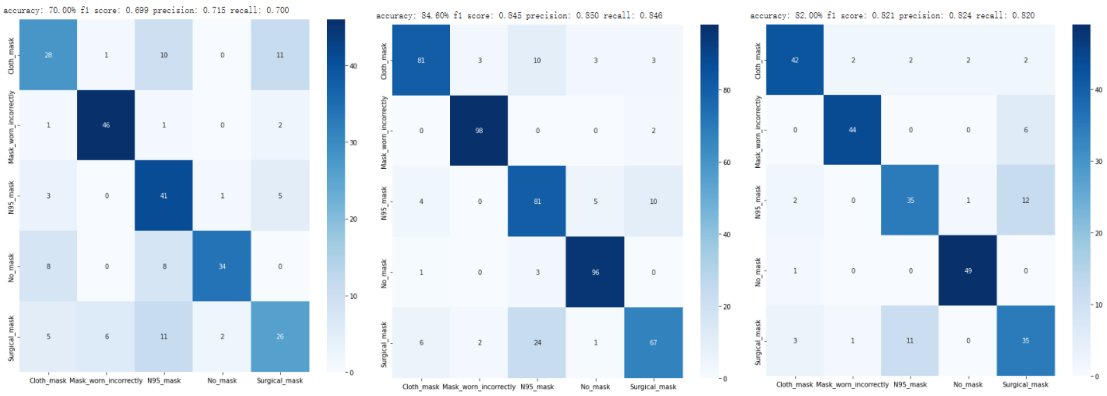

Figure 6: Confusion matrices for each age group: children(left), adult(middle) and old(right)

To test the gender bias of the CNN model, we test with 500 female images (100 images in 5 types of mask classes respectively) and 500 male images respectively. By comparing accuracy in Table.4, we can see that the accuracy of the model seems to have only changed a little bit. Due to the similar numbers of male and female images in the original dataset, we can infer that gender does not seem to have much effect on the model.

Table 4: performance of model on different genders

	accuracy	f1 score	precision	recall
female	83.2%	0.831	0.833	0.832
male	82.8%	0.829	0.832	0.828

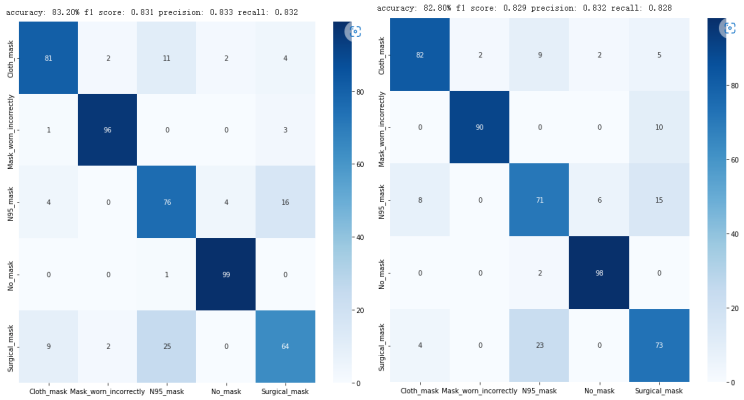

Figure 7: Confusion matrices for each gender:female(left) and male(right)

To ease bias among ages, we decide to use re-balance our dataset based on age groups. The new dataset contains more child images including all five mask classes. Due to the lack of number of child images in the previous dataset, we generate more data samples by using image augmentation, including horizontal flipping, brightness changing and resizing methods. Thus, after adding more image data, the new dataset now contains 2000 images, with 50 more child images and 50 more old people images for each mask class. After rebalancing the dataset, we retrain our previous model based on new dataset. The performance of our new model is shown as follows.

Table 5: performance of new trained model among different age groups and genders

	accuracy	f1 score	precision	recall
test data set	79.6%	0.796	0.809	0.793
children	72.8%	0.728	0.741	0.728
adult	81.4%	0.814	0.824	0.814
old	83.0%	0.831	0.837	0.830
female	82.8%	0.828	0.834	0.828
male	81.2%	0.813	0.824	0.812

As shown in Table.5, the new trained model still show no bias on gender category, as the results of female images and male images keep similar. With the new re-balanced dataset, the age categories show better results for child and old people groups , although the child group still gives a relatively low accuracy compared with the other one. The confusion matrices for our new models are shown as follows:

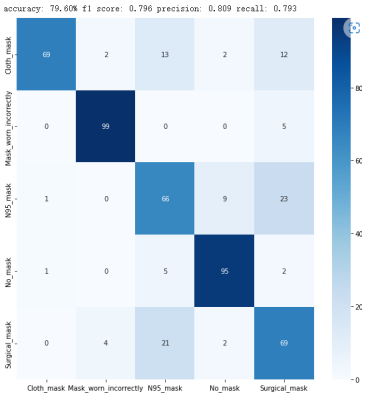

Figure 8: Confusion matrices for new trained model on test dataset

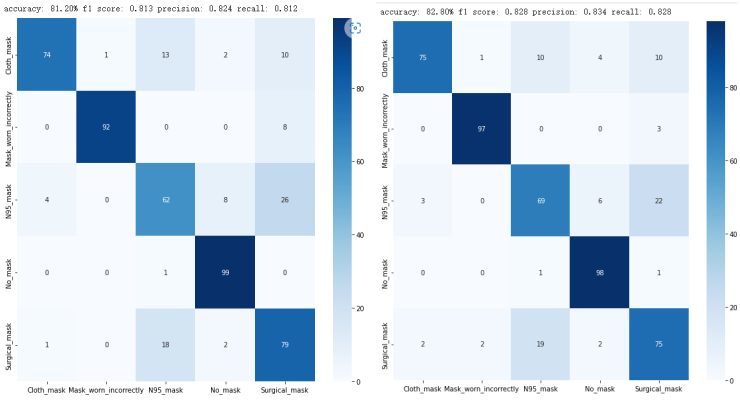

Figure 9: Confusion matrices for new trained model for each gender: male(left), female(right)

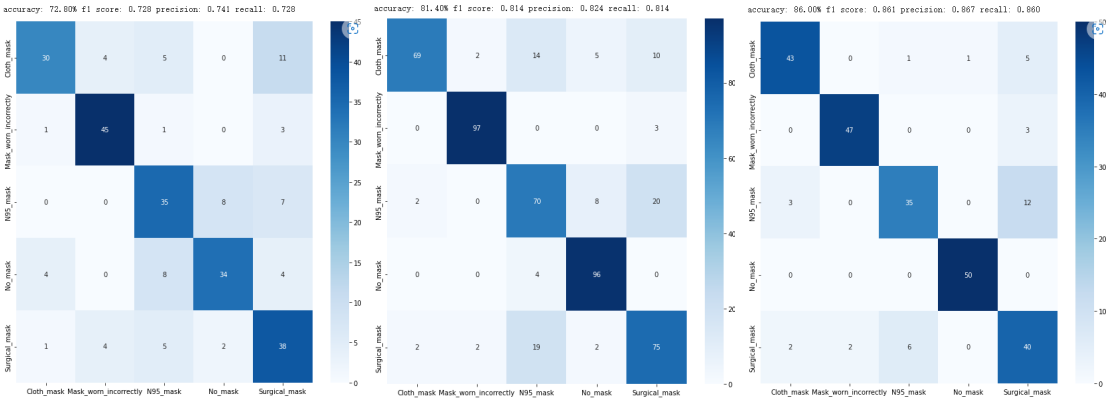

Figure 10: Confusion matrices for new trained model for each age group: child(left), adult(middle), old(right)

5 K-fold

Cross-validation is a statistical method used to evaluate and improve the performance of the machine learning models. It is commonly used in applied machine learning algorithms to compare and select a model for a given predictive modeling problem because it is easy to understand, simple to implement, and it generally gives model with a lower bias than the other methods.

Therefore, in order to help our model perform better accuracy, we will apply k-fold cross-validation to re-train the model. Instead of just simply splitting the data into training and validation/test sets based on the percentage, we shuffle our datasets and randomly split them according to the k value.

For each training progress, the dataset is going to split into k equal-sized fragments where $k-1$ of them are treated as training dataset and 1 piece is treated as validation dataset. In this project, we use funtion *StratifiedKFold*[2] provided by sklearn.model_selection. The function is able to automatically split the dataset k times and then we are able to iterate among the splitted datasets to do the training.

In the k-fold cross-validation, the k value must be chosen very carefully for the data samples. As k gets larger, the difference in size between the training set and the resampling subsets gets smaller. As this difference decreases, the bias of the technique becomes smaller. Which is, that there is a bias-variance trade-off associated with the choice of k value in the k-fold cross-validation. Here, we are going to use 10 as the number of k in this project, a value that has been found experimentally to give the model performance with a low bias and moderate variance.

Table 6: performance of model on each fold(original dataset)

	accuracy	f1 score	precision	recall
fold-0	80.5%	0.802528	0.805	0.814730
fold-1	80.5%	0.802155	0.805	0.802073
fold-2	81.0%	0.810778	0.810	0.812278
fold-3	77.5%	0.767173	0.775	0.781740
fold-4	84.0%	0.837518	0.840	0.838993
fold-5	78.5%	0.782428	0.785	0.785856
fold-6	80.0%	0.802856	0.800	0.814159
fold-7	79.0%	0.791516	0.790	0.800266
fold-8	81.0%	0.808135	0.810	0.811092
fold-9	80.0%	0.796688	0.800	0.805415
min	77.5%	0.767173	0.775	0.781740
max	84.0%	0.837518	0.840	0.838993
avg	80.2%	0.800178	0.802	0.806660

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html`

For the model trained with the original datasets, the performance metrics of our k-fold and aggregated data including min, max and average are listed in the following table.6. We can see that the accuracy of our model ranges from 77.5% and 84.5% where the accuracy(81%) of the model we trained in the first part of the project lies in this range. The averaged accuracy of our model trained by k-fold is 80.2%.

The performance of the model trained on re-balanced dataset is shown in Table.7. We can see that the minimum accuracy is 77.5% which is the same as the previous trained model and the maximum accuracy is 85.0% which increase 1% comparing to the model we trained on the original dataset. The accuracy of the model trained previously also lies in this range. For the averaged accuracy, it increases by 0.6 which means the sum of the accuracy of the newly trained model increases 6% in total as we have 10 folds. To summarize, there is actually not much improvements in the performance after we trained the new model on the new datasets.

Table 7: performance of model on each fold(re-balanced dataset)

	accuracy	f1 score	precision	recall
fold-0	82.0%	0.820409	0.820	0.830198
fold-1	85.0%	0.849421	0.850	0.851845
fold-2	78.0%	0.780712	0.780	0.783263
fold-3	80.5%	0.806044	0.805	0.828138
fold-4	79.5%	0.789999	0.795	0.796173
fold-5	83.0%	0.832036	0.830	0.844676
fold-6	78.0%	0.775632	0.780	0.792515
fold-7	83.5%	0.839352	0.835	0.849791
fold-8	77.5%	0.778220	0.775	0.798286
fold-9	81.0%	0.812907	0.810	0.825388
min	77.5%	0.775632	0.775	0.783263
max	85.0%	0.849421	0.850	0.851845
avg	80.8%	0.808473	0.808	0.820027

```
Epoch [4/5], Step [1/4], Loss: 0.6160, Accuracy: 79.60
Epoch [4/5], Step [2/4], Loss: 0.5384, Accuracy: 81.60
Epoch [4/5], Step [3/4], Loss: 0.5085, Accuracy: 81.20
Epoch [4/5], Step [4/4], Loss: 0.5238, Accuracy: 81.67
Epoch [5/5], Step [1/4], Loss: 0.5370, Accuracy: 80.40
Epoch [5/5], Step [2/4], Loss: 0.4913, Accuracy: 80.20
Epoch [5/5], Step [3/4], Loss: 0.4982, Accuracy: 81.20
Epoch [5/5], Step [4/4], Loss: 0.4744, Accuracy: 83.00

Epoch [4/5], Step [1/4], Loss: 0.5966, Accuracy: 77.20
Epoch [4/5], Step [2/4], Loss: 0.6102, Accuracy: 77.20
Epoch [4/5], Step [3/4], Loss: 0.6000, Accuracy: 79.20
Epoch [4/5], Step [4/4], Loss: 0.5769, Accuracy: 78.00
Epoch [5/5], Step [1/4], Loss: 0.4996, Accuracy: 83.00
Epoch [5/5], Step [2/4], Loss: 0.5801, Accuracy: 79.40
Epoch [5/5], Step [3/4], Loss: 0.5514, Accuracy: 80.60
Epoch [5/5], Step [4/4], Loss: 0.5257, Accuracy: 83.67
```

Figure 11: example of training accuracies during last two epoches

The recorded accuracy of these two models among 10 folds are shown in Fig.12. The accuracy keeps going up during training period. Examples of the training accuracy during last two epoches are shown in Fig,11 We can see that the training accuracy ends with just over 80% at last meaning that our model does not suffer from the overfitting problem.

In general, we can get to a conclusion that the performance of our model fluctuates based on how we split the dataset. However, the model has a relatively good performance overall and the model accuracy always holds above 77.5% and the highest performance of accuracy is 85%. Comparing the results we obtained for the original model (from Part I) using the cross-validation with our original model performance which used the single training/test split, we can see that the accuracy of the model we trained with single train-test split always lies between the minimum accuracy and maximum accuracy of the model that we trained with the cross-validation split. With train-test split, we do not care much about the splitting of the dataset which could affect much to the performance of a model. A model which do not works well could also has a good accuracy if it has a lucky initializing point. However, k-fold validation gives a better view on the performance of a model on the dataset. It provides more statistics and we can analysis the aggregated data including min, max, mean and so on to analysis the performance of a model. With more data points, the analysed performance of a model could be more reliable.

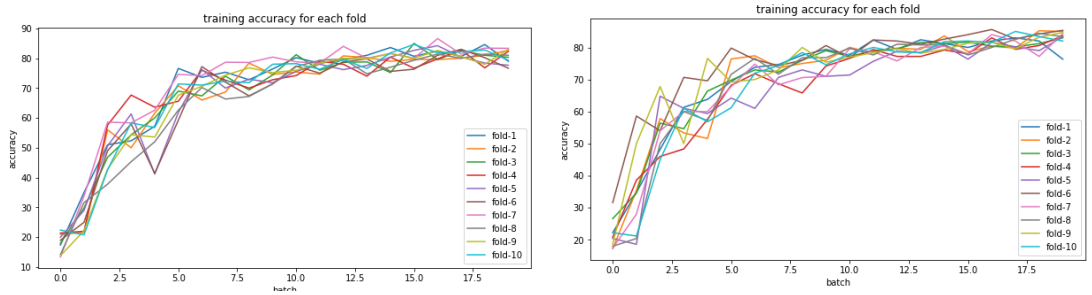

Figure 12: the change of training accuracy for 10 folds for model:
trained on original dataset(left) trained on re-balanced dataset(right)

References

[1] COVID Face Mask Detection Dataset, 07 2020. https://www.kaggle.com/datasets/prithwirajmitra/covid-face-mask-detection-dataset.

[2] Face Mask Detection, 05 2020. https://www.kaggle.com/datasets/andrewmvd/face-mask-detection?select=images.

[3] Face Mask Detection 12K Images Dataset, 05 2020. https://www.kaggle.com/datasets/ashishjangra27/face-mask-12k-images-dataset.

[4] Face Mask Detection Dataset, 06 2020. https://www.kaggle.com/datasets/wobotintelligence/face-mask-detection-dataset.

[5] FaceMask Dataset, 06 2020. https://www.kaggle.com/datasets/sumansid/facemask-dataset.

[6] COVID-19 Medical Face Mask Detection Dataset, 02 2021. https://www.kaggle.com/mloey1/medical-face-mask-detection-dataset.

[7] Adnane Cabani, Karim Hammoudi, Halim Benhabiles, and Mahmoud Melkemi. Maskedface-net – a dataset of correctly/incorrectly masked face images in the context of covid-19. *Smart Health*, 2020. http://www.sciencedirect.com/science/article/pii/S2352648320300362.