COMP 6721

Project assignment - Part 1

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

Muyi Chen(40202802) data processing

Junteng Ma(40204015) training

Zhixian Xiang(40017081) evaluation

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.

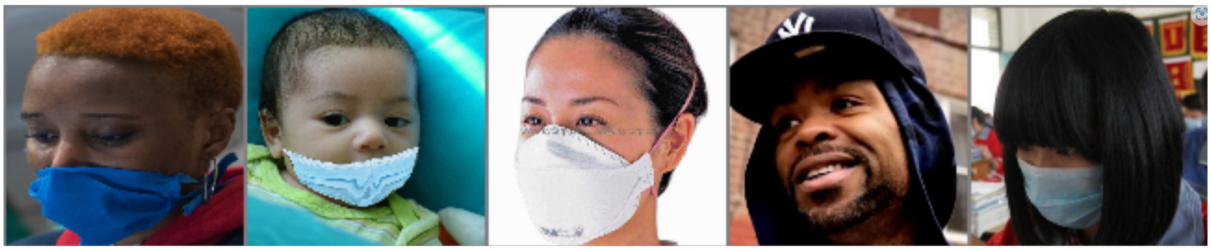
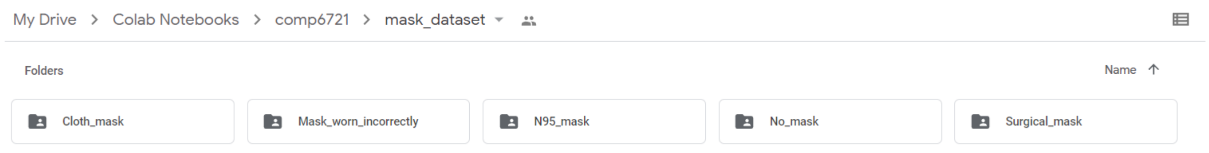


Figure 1: image samples from each category

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.



My Drive > Colab Notebooks > comp6721 > mask_dataset

Folders

Name ↑

Cloth_mask

Mask_worn_incorrectly

N95_mask

No_mask

Surgical_mask

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.

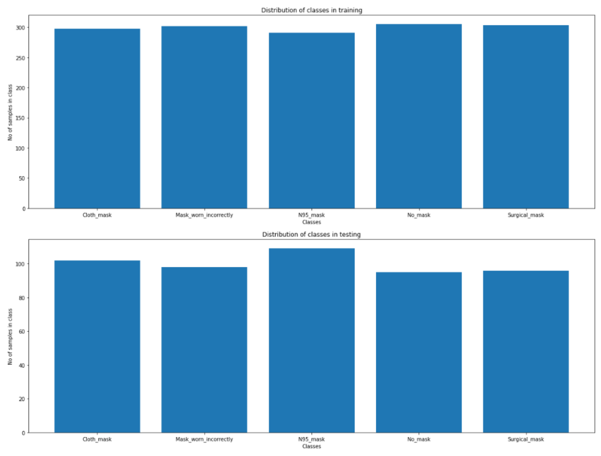


Figure 3: Number of data in training and testing dataset from each categories

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. $\mathrm{FC}-a$ 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

input (128x128 RGB image)		
CNN	**CNN with less pooling**	**deeper CNN**
conv3-64	conv3-64	conv3-64
maxpool2	conv5-128 (stride 2)	maxpool2
conv5-128		conv5-128
		maxpool2
		conv3-256
		conv3-256
AdaptiveAvgPool 7x7		
FC-250		
FC-5		
softmax		

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.

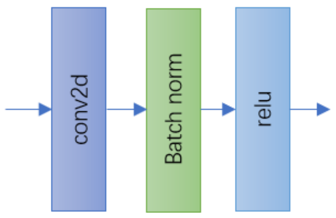


Figure 4: structure inside a CNN block

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%.

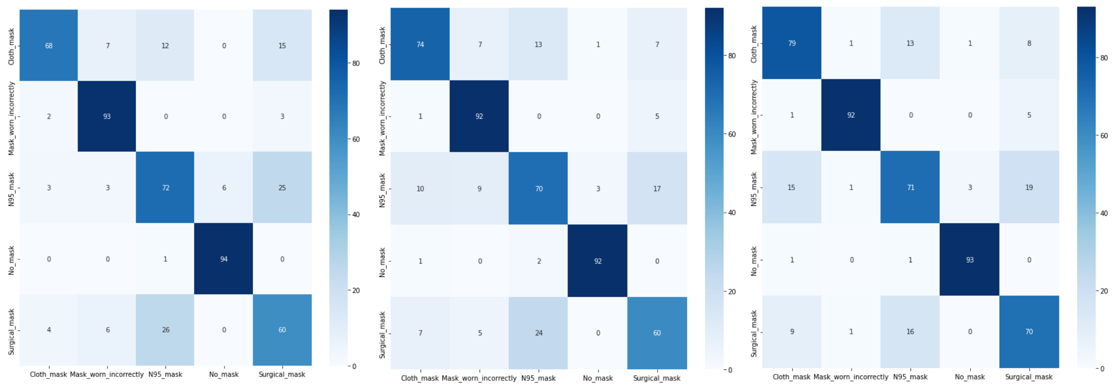


Figure 5: confusion matrix for each model
(left: normal CNN, center: CNN with less pooling layers, right: deeper CNN)

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.

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.