Project Title : Face Mask Detection

Students :

1. Name : Yang, Po-Yen
   Student ID : 20561878
   HKUST email address : pyangaf@connect.ust.hk
2. Name : Huang, Yi-Feng
   Student ID : 20470495
   HKUST email address : yhuangcr@connect.ust.hk

Description of the dataset and any preprocessing [10 points]

The dataset that we chose is the face mask detection.
(https://www.kaggle.com/andrewmvd/face-mask-detection)
The raw dataset contains a folder of images and a folder of xmls annotations. A xml has a label labeling its corresponding image and contains one or more “objects”, each “object” describes a human face in the image. The object is described with the bounding box of the human face in the image with “xmin”, “xmax”, “ymin”, “ymax”. And there are various labels associated with the object, including “pose”, “truncate”, “occluded”, and “difficult”.

For data preprocessing, we have done the following procedures:

1. Reading the notation on each “object” (head image) in the dataset from the xml file. For every object, extract the bounded head image from the raw image, and save the extracted head image into a new image in png format. Extract the labels associated with this object and write it to a dataframe and store it to a csv file. The csv file contains the following columns :
   - image_path : stores the path to the head images that we extracted from the original image provided in the dataset.
   - label : containing 0 (no face mask worn), 1 (face mask properly worn), and 2 (face mask worn incorrectly).
   - pose : unspecified
   - truncated : 0
   - occluded : 0
   - difficult : 0

2. When loading the preprocessed cropped image from the data, since the head images that we saved are in png format, which has 4 channels, we converted those images to RGB format, which contains only 3 channels.

3. We also performed some data augmentation methods :
   - Resize image : resize the head images to 32*32 so that every image has identical image size.
   - Flip the image : perform a horizontal flip on the images randomly.
   - Rotate the image : perform a 15 degrees rotation on the images randomly.
4. Convert the images to Tensor

5. Split the extracted head images into training set, validation set, and test set with proportion 70 : 20 : 10.

Description of the machine learning task performed on the dataset [10 points]

Our machine learning task is that given an image input of a human head, the output will be the feature “label” that classifies the image into 3 classes: “no face mask worn”, which is labelled as 0, “face mask properly worn”, which is labelled as 1, and “mask worn worn incorrectly”, which is labelled as 2.

In the training process, we trained the model to predict the feature “label” based on the head image, which from the raw data input is the head image bounded by the “xmin”, “xmax”, “ymin”, “ymax”.

There are 4 other labels in the photo. For “pose”, “truncated”, “occluded”, “difficult”, by running some statistics with code, we observed that all data entries in “pose” are labeled as unspecified, also, “truncated”, “occluded”, and “difficult” are all labeled as 0. Hence, they are omitted in the training process.

Description of the hardware and software computing environment, machine learning methods, and parameter settings [10 points]

For this project, we use Google Colab for hardware.
As for the software part, we have utilized the following softwares :

1. PyTorch : version 1.8.1
2. Numpy : version 1.19.5
3. Pandas : version 1.1.5
4. Image (from PIL) : version 7.1.2
5. torchsummaryX : version 1.3.0
6. tqdm : version 4.41.1
7. torchvision: version 0.9.1+cu101
8. sklearn: version 0.24.1
9. seaborn: version 0.11.1
10. matplotlib: version 3.2.2

For machine learning methods, we chose to solve the problem with CNN. We think this is a suitable method as the inputs are images. And CNN has been an effective method to deal with image inputs as it can learn through the relationship between the pattern in images and generating an output.
For the output, the CNN will give out three scores for the label 0, 1 and 2. The highest score among the three will be the output label. In addition, cross entropy loss is good at solving

classification problems. Hence, for our project, CNN will learn to give the correct output with cross entropy loss.

Parameter settings

1. Our own-built CNN model :

Layer	Kernel size	# of kernels	Stride	Padding
Input	3 x 32 x 32 image			
Conv	3 x 3	32	1	1
Max pooling	2 x 2	-	2	0
Conv	3 x 3	64	1	1
Conv	3 x 3	128	1	1
Max pooling	2 x 2	-	2	0
Conv	3 x 3	256	1	1
Conv	3 x 3	512	1	1
Average pooling	8 x 8	-	1	0
Output	1 x 512 vector			

   For every convolutional layer, it is followed by a batch normalization and a ReLU activation.
2. Fully connected layers :
   - 2 fully connected layers
     - 512 hidden layers in the first layer (512 x 512), followed by a ReLU activation function
     - 3 output layers in the second layer (512 x 3), followed by a ReLU activation function
   - Apply the sigmoid function
3. Fully connected layers with dropout :
   - 2 fully connected layers and 1 dropout layer
     - 512 hidden layers in the first layer (512 x 512), followed by a ReLU activation function
     - Dropout layer with dropout probability = 0.1
     - 3 output layers in the second layer (512 x 3), followed by a ReLU activation function
   - Apply the sigmoid function
4. Training Parameters:
   20 epochs
   Criterion : CrossEntropyLoss
   optimizer : Adam (with default settings)

Source code adhering to good programming practices [10 points]

We have adapted part of our code from the source :
https://www.kaggle.com/nageshsingh/mask-and-social-distancing-detection-using-vgg19

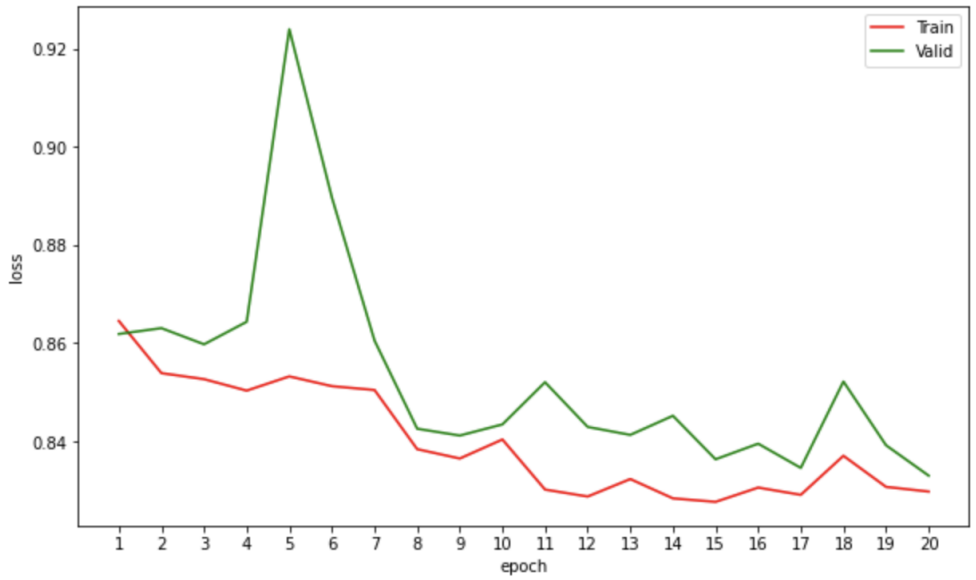
Description of the experiments [25 points]

1. Own-built CNN model (baseline model)
    - 20 epochs
    - Criterion : CrossEntropyLoss
    - optimizer : Adam (with default settings)
    - model : Own-built CNN model + Fully connected layers
2. Model improvement experiments
    - Resnet 50
        - 20 epochs
        - Criterion : CrossEntropyLoss
        - optimizer : Adam (with default settings)
        - model : resNet50
    - Changing the learning rate of the Adam function
        - 20 epochs
        - Criterion : CrossEntropyLoss
        - optimizer : Adam (learning rate = 0.0001)
        - model : own-built CNN model + Fully connected layers
    - Changing the number of epochs and the learning rate of the Adam function
        - 50 epochs
        - Criterion : CrossEntropyLoss
        - optimizer : Adam (learning rate = 0.0001)
        - model : own-built CNN model + Fully connected layers
    - Adding dropout layer
        - 50 epochs
        - Criterion : CrossEntropyLoss
        - optimizer : Adam (with default settings)
        - model : own-built CNN model + Fully connected layers with dropout

Visualization and discussion of the results obtained [20 points]

1. Graphs for training loss and validation loss, with validation and test accuracy
    - Own-built CNN model (baseline model)

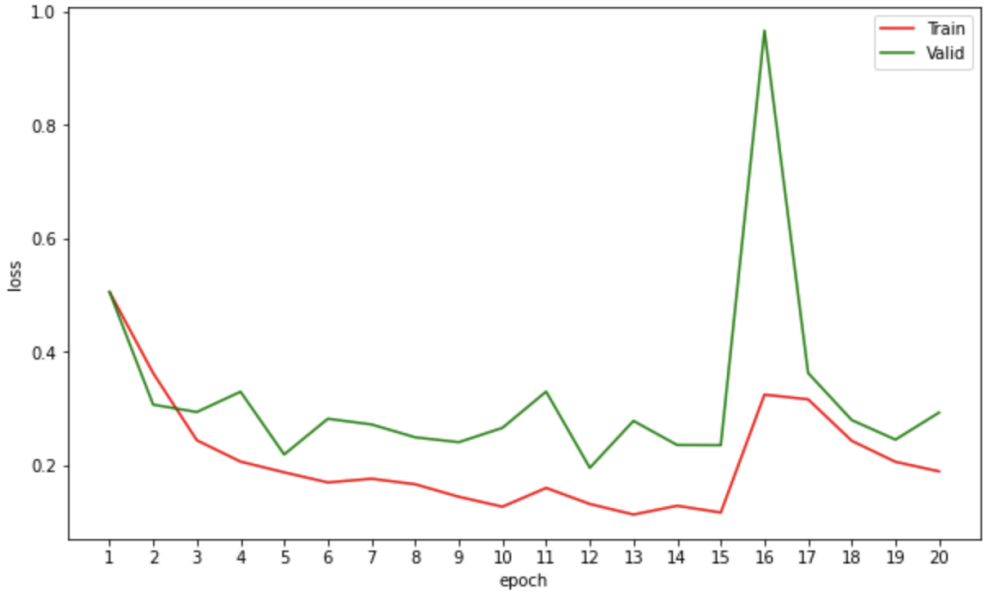
        Train Loss: 0.8298 Valid Loss: 0.8330
        Valid Accuracy: 94.00% Test Accuracy: 93.40%



- Resnet 50

  Train Loss: 0.1671 Valid Loss: 0.2497

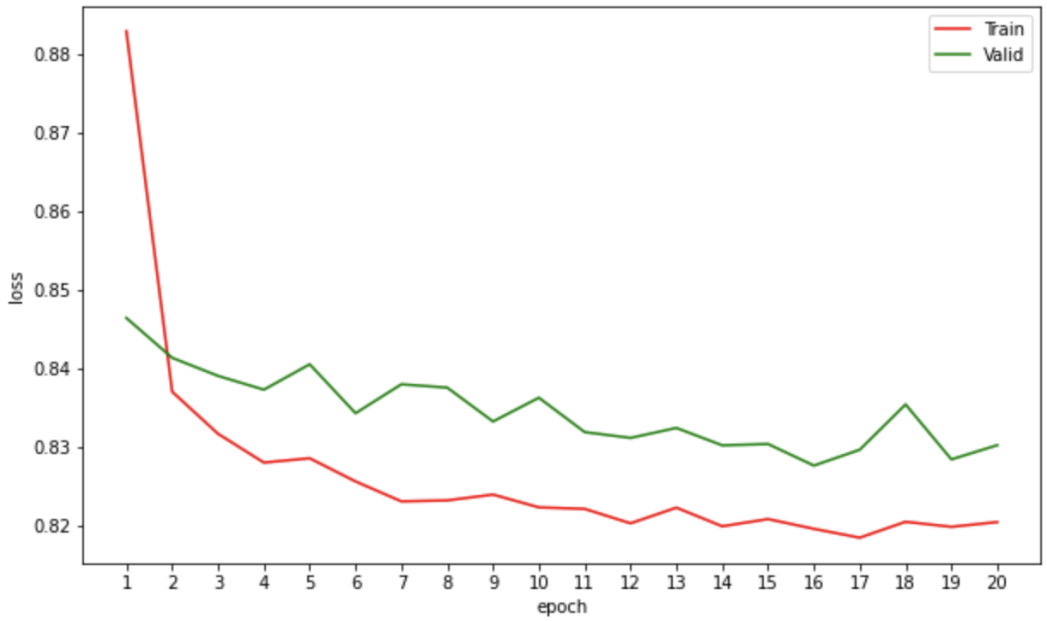
  Valid Accuracy: 94.61% Test Accuracy: 93.89%



- Changing the learning rate of the Adam function

  Train Loss: 0.8195 Valid Loss: 0.8276

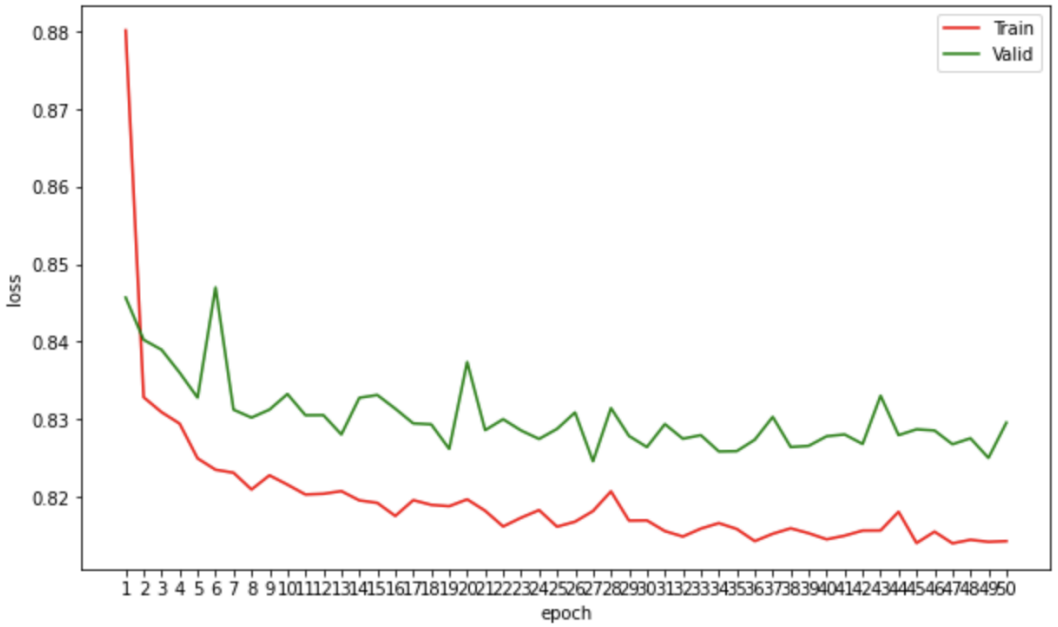
  Valid Accuracy: 95.47% Test Accuracy: 94.62%



- Changing the number of epochs and the learning rate of the Adam function

  Train Loss: 0.8181 Valid Loss: 0.8246

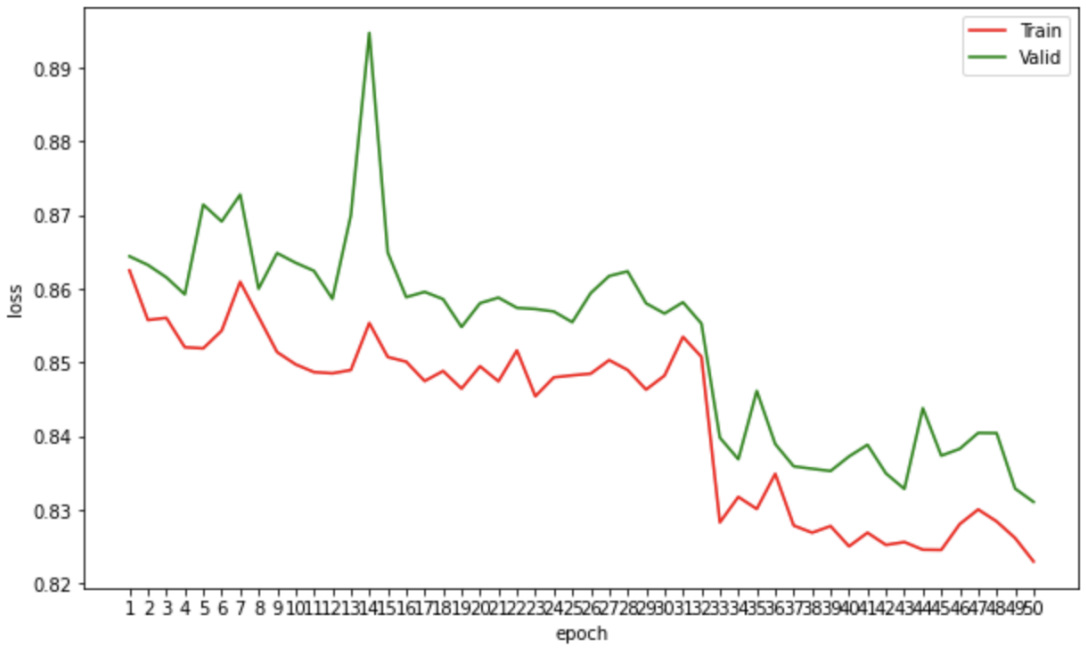
  Valid Accuracy: 95.96% Test Accuracy: 93.64%



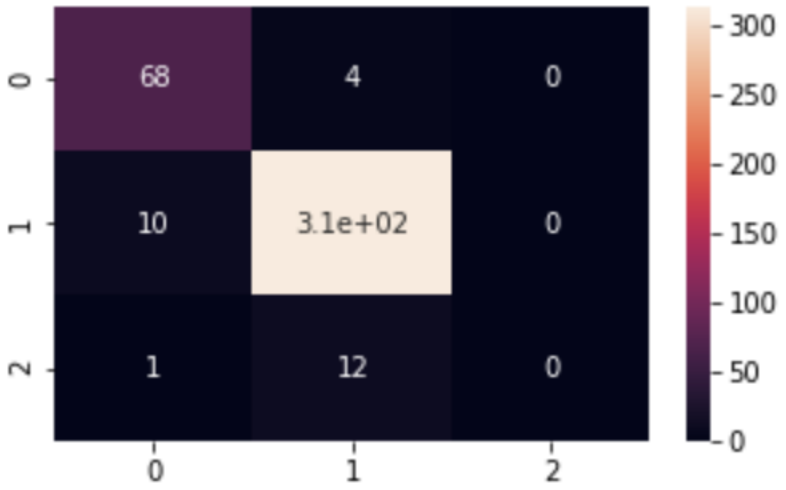
- Adding dropout layer

  Train Loss: 0.8256 Valid Loss: 0.8328

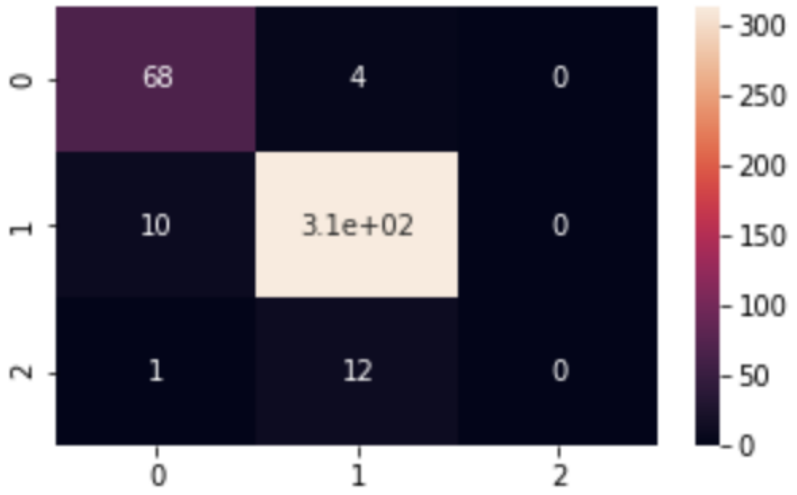
  Valid Accuracy: 94.24% Test Accuracy: 94.13%



2. Confusion matrix
    - Own-built CNN model (baseline model)

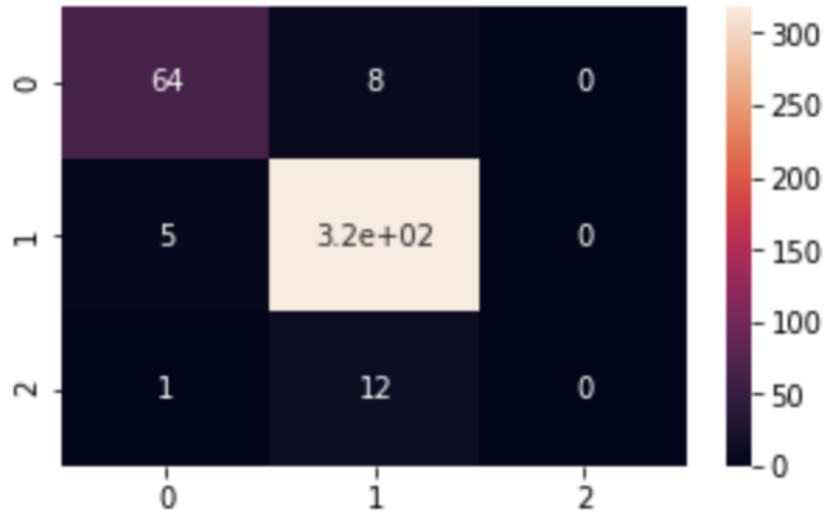


    - Resnet 50



- Changing the learning rate of the Adam function

- Changing the number of epochs and the learning rate of the Adam function



- Adding dropout layer

3. Discussion
    - The validation accuracy and test accuracy for resNet is slightly higher than that of our baseline model, this is aligned with our expectations, since resNet has more hidden layers than our baseline model, it should perform better with a higher accuracy.
    - The validation accuracy and test accuracy when we changed the learning rate to 0.0001, which is smaller than the default setting, is higher than that of our baseline model, this is also aligned with our expectations. As our baseline model has a high accuracy, we think that by decreasing the learning rate would allow the model to get closer to the optium as the step is smaller, and the result also proves our thought.
    - The validation accuracy and test accuracy when we changed the number of epochs to 50, and learning rate to 0.0001, which is smaller than the default setting, is higher than that of our baseline model. However, the test accuracy is lower than that of the model where the learning rate is 0.0001 and the number of epochs is 20, and this is also aligned with our expectations. This is because when the number of epochs is 20, the model has already shown a pattern of convergence, hence, by increasing the number of epochs, the validation accuracy has not increased, but the test accuracy has decreased as we think it is becoming overfitting.
    - When adding the dropout layer, the validation accuracy and test accuracy has slightly increased compared to the baseline model. We think the reason may be that the number of our data is not really large, and by having the dropout layer, the number of data used for training and validation has decreased. hence, the validation accuracy increased as it has less probability of getting an error. It can also be observed that the loss function decreases in a training epoch as dropping out nodes slows down the immediate adoption of the model towards the dataset but allows the model to be more robust.
    - The number of label “2” data in the dataset is really little, hence, we think that this is the reason why only label “0” and label “1” have occured as the prediction labels for the test set, and there are no images predicted as label “2”.

Division of labor

- Yang, Po-Yen (50%)
    - Data preprocessing (transform, data augmentation, and create data loader)
    - Build CNN model and compared models
    - Build model training functions and perform test set prediction
    - Perform all the model experiments (baseline model + 4 model experimentations)
    - Report writing
    - Video recording
- Huang, Yi-Feng (50%)
    - Data preprocessing (Extracting image and csv from raw kaggle archive zip)
    - Train function/ test function writing and refactoring
    - Exploring potential improvement (Dropout layer)
    - Data visualization
    - Report writing
    - Video recording

Video presentation (expected to cover all of the above) [15 points]
https://youtu.be/l7eCPv1gwXA