Concordia University

Engineering and Computer Science

COMP 6721
Applied Artificial Intelligence

Project Part 1: AI Face Mask Detector

Submitted To: Dr. Renne Vitte

Contributors

Jhanvi Arora (40162512)

Manish Sehgal (40165366)

Zalakben Rajendrakumar Patel (40164315)

CONTENTS

Team Member Specialization:

Jhanvi Arora

- Contributed to the training process in terms of automation and refactoring.
- Derived valid metrics to judge the efficiency of model and proposed changes to the existing model to derive the required metrics

Manish Sehgal

- Proposed the initial training model to fit the on-hand dataset.
- Inferring from the retrieved metrics, tested multiple model configurations making changes in base model.

Zalakben Rajendrakumar Patel

- Retrieved the parts of the dataset from various sources, merged and categorised them into required categories.
- Applied the pre-processing strategies to make data uniform.

Dataset

For machine learning models to perform different actions, it is essential to feed relevant training dataset. Along with training sets, testing datasets are taken into consideration to guarantee that the model is interpreting the data correctly. In this project, we have taken 2008 images in total, which consists of a training set containing 1509 images and a testing set containing 499 testing and validation images. Below table shows distribution of images into 5 categories; Cloth mask, surgical mask, ffp2, ffp2 with valve and without mask.

1.1 Scraped Data

Initially, due to absence of any categorically defined mask dataset, we scraped the images from the internet using download all images chrome extension [7]. All the scraped images are not relevant, hence the relevant images had to be filtered manually amongst them. It did give us sufficient categorical data although it does contain a lot of class invariance.

1.2 Defined Dataset

Some classes such as FFP2 with valve and FFP2 in existing data had a lot less images as compared to the other readily available classes like the Surgical or Cloth mask, therefore we decided to extract data from readily available datasets on Kaggle [1], but we had to manually go through images and find the required class data. We did collect a few images from the aforementioned datasets and categorized them in 5 categories. The selected images are from Kaggle‘s 3 Face Mask Datasets; [2] [3] [4]. A few images from these sets were dropped due to multiple face issues and bad resolution.

1.3 Data Pre-processing

To convert the images in model acceptable format, they are converted and resized to a uniform size of 64*64 across the whole dataset. The input images are then organized and transformed to batch tensors to provide for the model input.

1.4 Data Splitting

Following the model’s training requirements - the data is initially split into three folders of train, test and validation. The train data maintained at 75% of the total data consist of over 1509 images and the other two sets contain a total of 499 images.

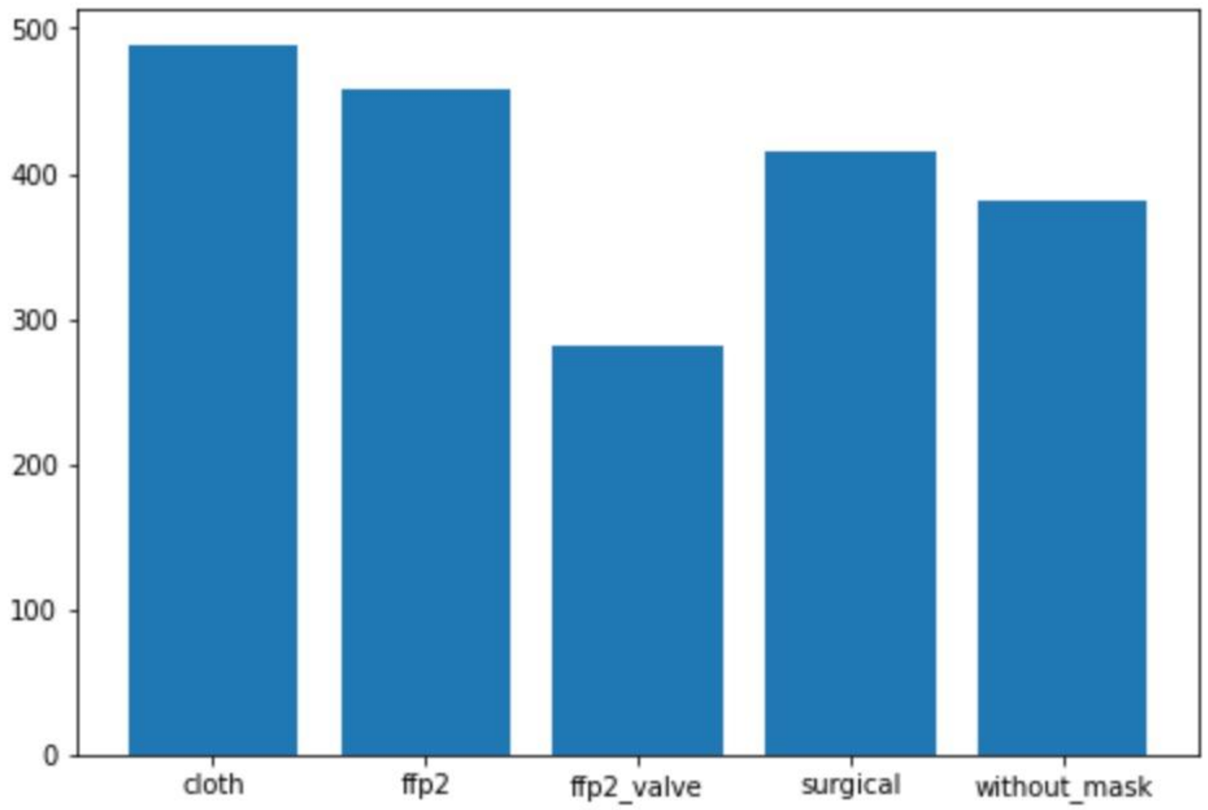


Figure 1: Dataset statistics

CNN Architecture

Deriving from the requirements from the project, Convolutional Neural Network is trained and used for deducing the class of the mask. Face mask training images fed here contain a series of pixels, where pixel values signify about brightness and colour of the pixel. The applied CNN Architecture here comprises three layers: Convolutional Layer, Pooling Layer and Fully connected layer.

Since the convolutional layer is the core building block contributing to major computational load, we have trained our model using 4 convolutional layers and 3 max pool layers. In convolutional layer, the dot product is computed between a matrix of a restricted portion of our image and a matrix containing a set of learnable parameters, i.e. Kernel. Convolutional layer is followed by ReLU as an activation function. After all convolutional layers, features are flattened and are furthermore connected to a fully connected unit containing 512 neutrons that in forward propagations connects to a 256 unit hidden layer and then narrows to generate a resultant layer of 5 neurons. Step by step executions of the CNN architecture are described below.

1. We have taken input of images with dimension 64*64. The first convolutional layer has filter size 3, which is the same in the following convolutional layer as well. Input channels are defined to be 3 since the image is an RGB and the corresponding number of filters applied to it are 32. It uses 2 strides and 1 padding. To uphold the image resolution, a padding of size 1 is also included.
2. After the first convolutional layer, activation function ReLU is applied, which is followed by the second convolutional layer having 32 input channels and producing 64 output channels. This convolutional layer uses 1 stride and 1 padding.
3. Before applying third convolutional layer, 2*2 max pool layer is used to slowly reduce the size of learning filters formed. To normalize the input size at each layer, we have added batch normalization after max pooling.
4. Third convolutional layer takes 64 input channels and produces 128 output channels. This convolutional layer uses 1 stride and 1 padding. It is again followed by 2*2 max pool layer and batch normalization.
5. The Fourth convolutional layer takes 128 input channels and produces 256 output channels, which is followed by activation, max pooling and batch normalization.
6. Finally filters are flattened to convert our data into a 1-dimensional feature input, which is followed by linear transformation to create a single layer feed forward network.

7. The flattened features connect to a linear layer of 512 neuron units.
8. Another hidden layer of 256 units connects to the previous hidden layer with a dropout of 0.5 and then in forward network generates the output of 5 class units.
9. Learning rate of our model in training is kept 0.0001. We have used a total of 15 epochs in order to train our model. Epochs can be increased in order to achieve better performance, but it may lead to overfitting at times.

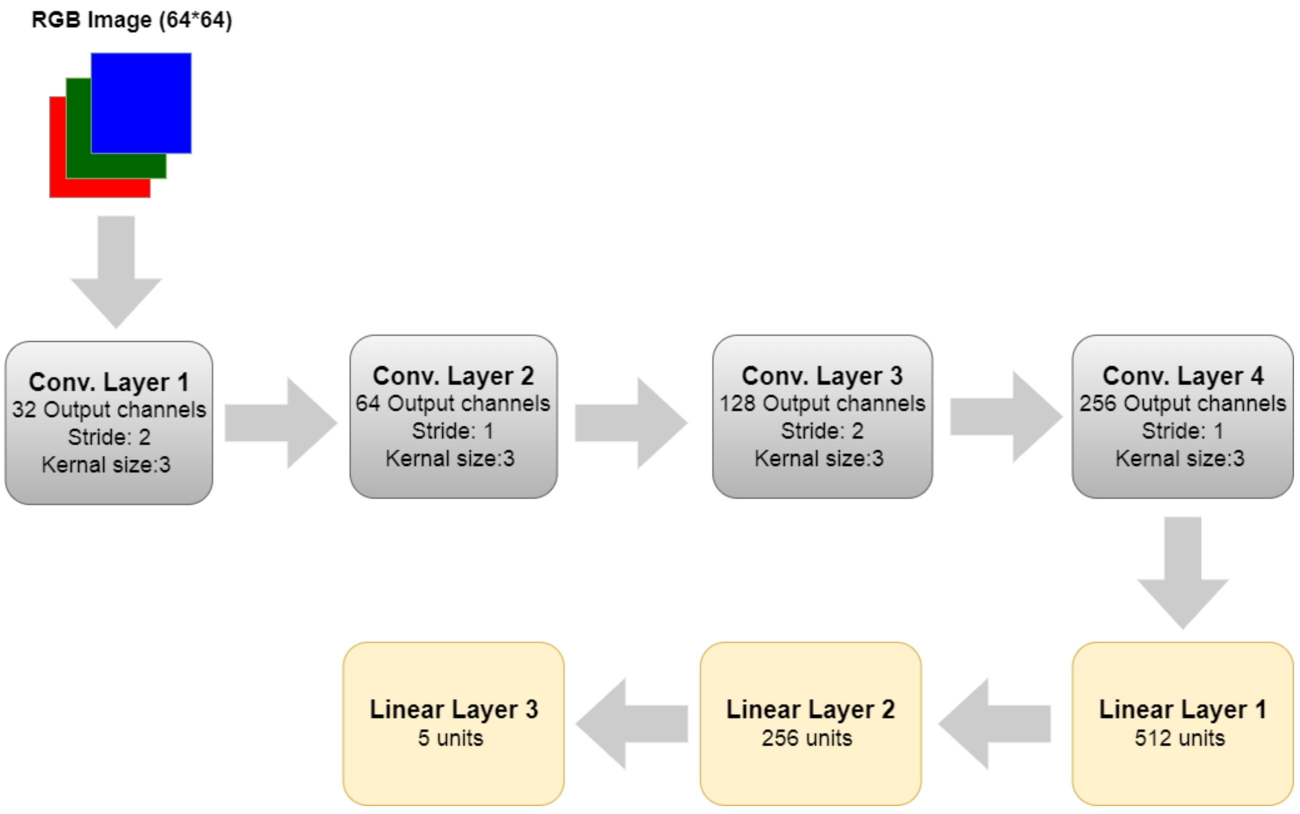
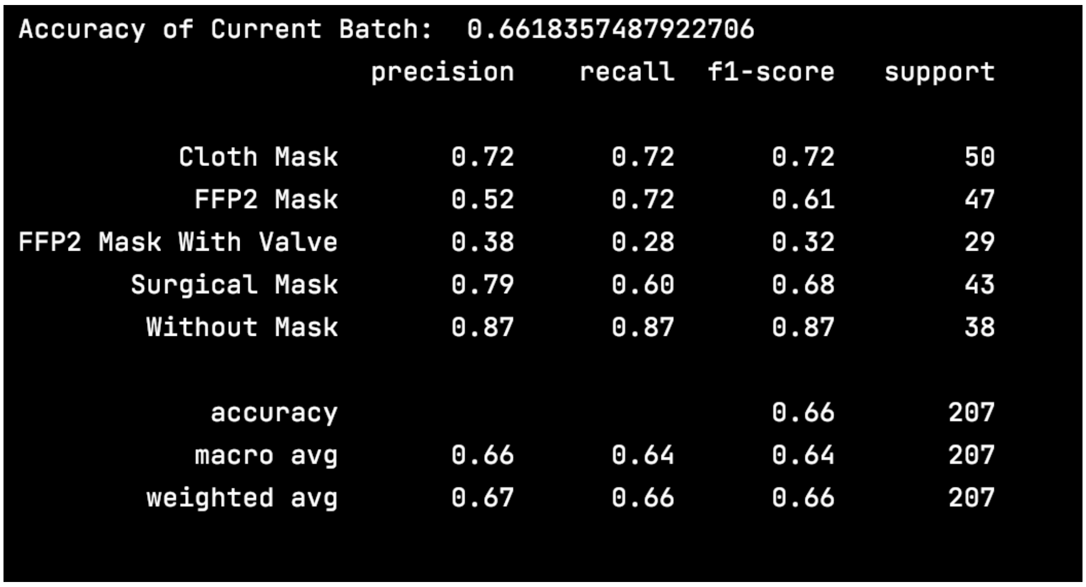


Figure 2: CNN Architecture

Evaluations

Remarks Before Testing Data

- Image data provided in the project is split into 3 subsets: Training set, Validation set and Test set. 75% of the data is used as a training set, the rest 25% data is used in testing and validation.
- Training data is taken in equal proportions from each labelled class to avoid wrong classifications and to improve the accuracy.
- We have used existing library split_folders [8] to make these almost equal splits.
- Below classification matrix generated through implementations provides a record of precision, recall, accuracy and F1-score for all 5 classes separately.
- Our model is currently achieving 78% training accuracy and 66% validation accuracy.



```
Accuracy of Current Batch:  0.6618357487922706
                      precision    recall  f1-score   support

          Cloth Mask       0.72      0.72      0.72        50
           FFP2 Mask       0.52      0.72      0.61        47
FFP2 Mask With Valve       0.38      0.28      0.32        29
        Surgical Mask      0.79      0.60      0.68        43
        Without Mask       0.87      0.87      0.87        38

            accuracy                           0.66       207
           macro avg       0.66      0.64      0.64       207
        weighted avg       0.67      0.66      0.66       207
```

Figure 3: Performance Matrix

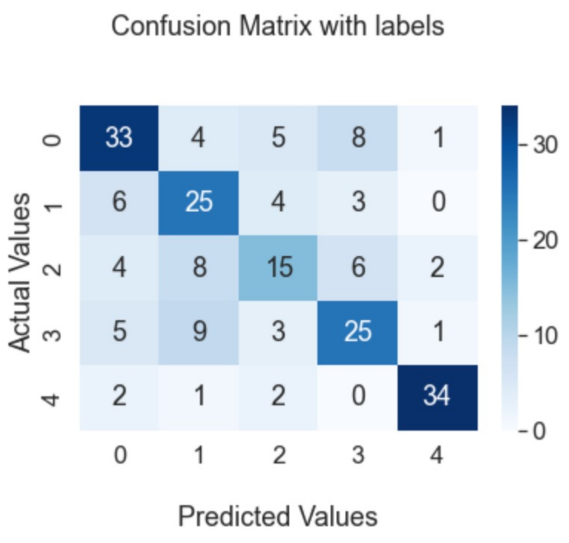


Figure 4: Confusion Matrix

Observations:

- This accuracy can certainly be improved with proper weight initialization and better parameter settings. This accuracy is achieved with 10 epochs.
- Certainly the lack of more images in the FFP2 with valve class does reflect poorly on its f1-score. Also, the quality and resolution of images do not lead to proper formation of ROI in the images- such as in the valve of FFP2 images.
- Increasing number of epochs along with addition/modifications of several convolutional and hidden layers can also lead to significant increase in accuracy once data is properly augmented.

REFERENCES

[1] https://www.kaggle.com/datasets/andrewmvd/face-mask-detection

[2] https://www.kaggle.com/datasets/ashishjangra27/face-mask-12k-images-dataset

[3] https://www.kaggle.com/aditya276/face-mask-dataset-yolo-format

[4] https://www.kaggle.com/datasets/prithwirajmitra/covid-face-mask-detection-dataset

[5] https://www.kaggle.com/datasets/omkargurav/face-mask-dataset?select=data

[6] https://www.kaggle.com/datasets/vijaykumar1799/face-mask-detection

[7]
https://chrome.google.com/webstore/detail/download-all-images/ifipmflagepipjokmbdecpmj
bibjnakm/reviews

[8] https://pypi.org/project/split-folders/

APPENDIX

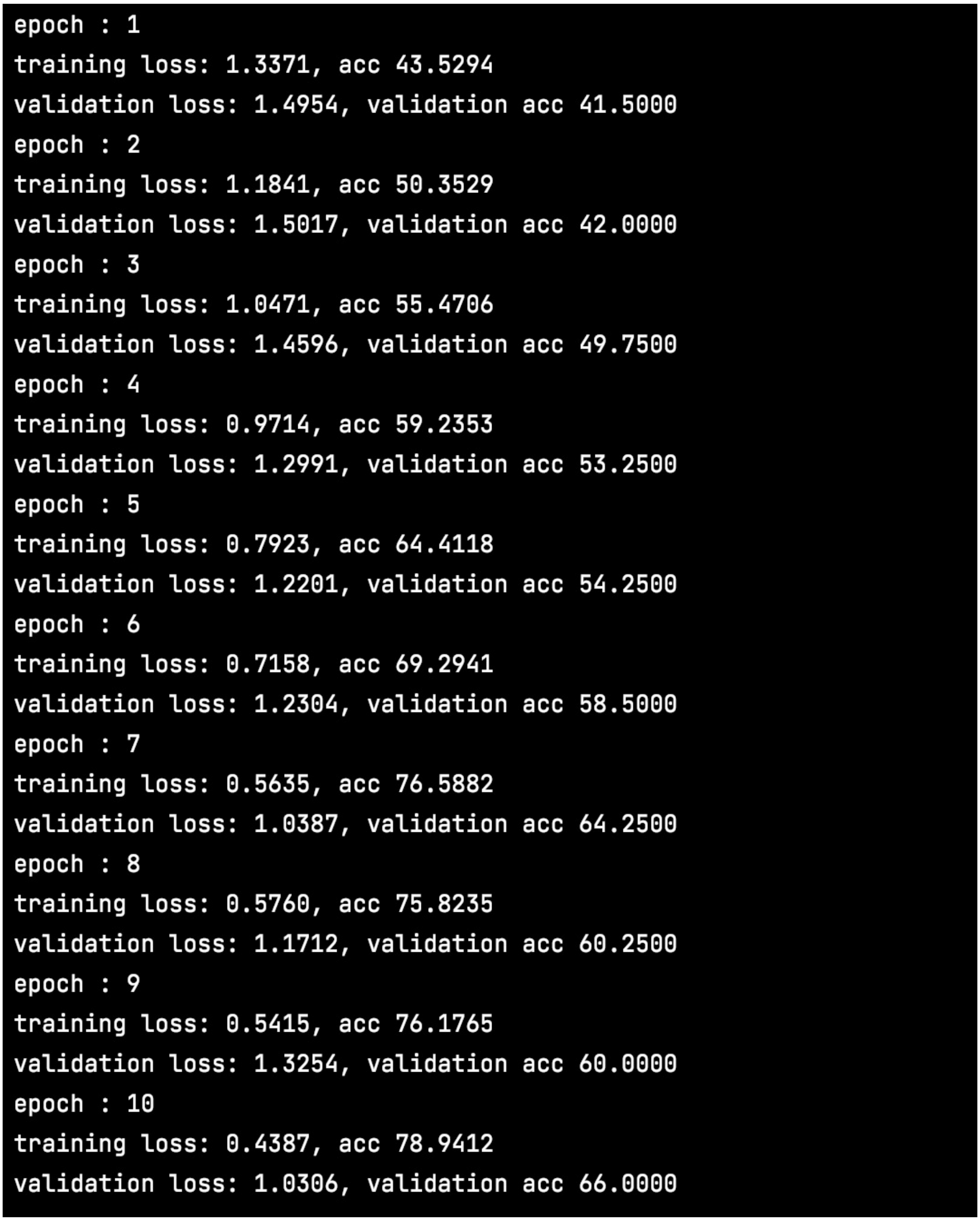

```
epoch : 1
training loss: 1.3371, acc 43.5294
validation loss: 1.4954, validation acc 41.5000
epoch : 2
training loss: 1.1841, acc 50.3529
validation loss: 1.5017, validation acc 42.0000
epoch : 3
training loss: 1.0471, acc 55.4706
validation loss: 1.4596, validation acc 49.7500
epoch : 4
training loss: 0.9714, acc 59.2353
validation loss: 1.2991, validation acc 53.2500
epoch : 5
training loss: 0.7923, acc 64.4118
validation loss: 1.2201, validation acc 54.2500
epoch : 6
training loss: 0.7158, acc 69.2941
validation loss: 1.2304, validation acc 58.5000
epoch : 7
training loss: 0.5635, acc 76.5882
validation loss: 1.0387, validation acc 64.2500
epoch : 8
training loss: 0.5760, acc 75.8235
validation loss: 1.1712, validation acc 60.2500
epoch : 9
training loss: 0.5415, acc 76.1765
validation loss: 1.3254, validation acc 60.0000
epoch : 10
training loss: 0.4387, acc 78.9412
validation loss: 1.0306, validation acc 66.0000
```

Figure 5: Training Model Matrix