COMP 472 - Project 1 Report

Group NS_07
Jason Dinh (40129138) - Data Specialist
Axel Dzeukou (40089940) - Training Specialist
Vyacheslav Medvedenko (40134207) - Evaluation Specialist
Dante Di Domenico (40125704) - Compliance Specialist

Github Repository: https://github.com/jasondinh99/COMP472_Project

Dataset

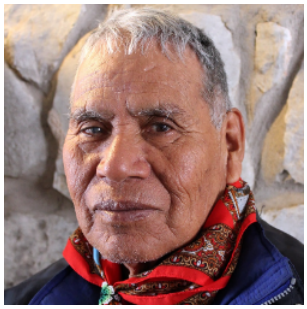
Developing an AI can be difficult at times, but what is the most crucial within the process is choosing the right data set when needing to learn how to decipher images. In this scenario, this AI is being taught to analyze images of people's faces, the purpose is to determine not only if the individual is wearing a mask or not, but what specific mask they are wearing as well, whether it be a cloth mask, procedural mask, or any of the FFP2/N95/KN95 masks.

For this project, numerous images of individuals are needed to be collected by the AI and deciphered into their respective categories, which from top left to bottom right [see Figure 1] reads no mask, cloth-material mask, surgical mask, and particulate respirator mask such as the N95. For the AI to have sufficient material to train with, over 1600 photos have been collected from Kaggle.

For each class, the specific number of photos are as shown:

- 420 no mask images. Sources: [1], [3]
- 402 cloth mask images. Sources: [1], [2], [3]
- 400 surgical mask images. Sources: [1], [2], [3]
- 410 N95 mask images. Sources: [1], [2], [3] [4]

This amount totals to 1632 photos of individuals.



No mask

Cloth mask

Surgical mask

N95 mask

Figure 1: Categories of images

Just so that the dataset would have some distinction to its photos, the angles of which they were taken vary from close-ups, side profiles and portraits of individual's faces. Each image contains only one fully visible face and photos of groups of people were not taken into account as they were not necessary. To avoid any complications when implementing the photos into the program, size formatting was needed with almost all the photos, as well as manually sorting them into each of their designated categories. As it would be easier to maintain a consistent size for the photos, many were cropped to a 1:1 ratio.

As for the process, not all the photos were used in both the training and testing. Around 1200 were used within the training segment while approximately 400 were used for the testing segment, with an equal number from each category.

IMPORTANT: Trained model

We cannot include our trained model in the Moodle submission and on Git because the size is too large for both (350+ MB). To use our trained model, please download the model from our Google Drive and save it in the root folder (the same folder as the 'main.py' file).

Link: https://drive.google.com/file/d/1RteV9Hwqbqr8MRca4xAm5OW3FatnXcCT/view?usp=sharing

CNN Architecture

In the beginning, the datasets of images are all reshaped into size 150 x 150, transformed into tensors, and then converted into dataloaders. For the process of testing and training to pass quicker, shuffling was set to true such that the training data is reshuffled at every epoch while using 4 workers.

From here, the ‘fit’ function is run, which in turn trains the model, prints the validation accuracy for every epoch, and saves the model only if the validation accuracy is of at least 60%.

To extend the functionality of “torch.nn.Module”, a base class is created (the base class used to develop all neural networks). We then add various functionalities to the base class to train and validate the model, as well as get the result for each epoch.

Our model is inspired by reference [6], which will be initiated from the “FaceMaskClassifier” class which inherits from the base class. Within this model, there are 3 CNN blocks, each consisting of 2 convolution layers and 1 max-pooling layer. While stride and padding are set to 1, the Relu activation function is used so that negative values are removed from the feature map as pixel values cannot be within the negative range of integers.

Once convolution has been applied and features from the image have been extracted, a flatten layer is used to flat the tensor. The tensor contains three dimensions to begin with, but after the flatten layer has undergone its job, it leaves the converted tensor to be one-dimensional. Once complete, three linears are added to reduce the size of the tensor and learn its features.

At the end of the training process, the saved model is then reloaded so that it may provide us with the predicted labels of our testing set. The predicted labels are classified as either “0”, “1”, “2”, or “3” which stand for “cloth_mask”, ”n95_mask”, ”no_mask”, ”surgical_mask” respectively.

Ideally we were expecting a model that gives us a validation accuracy of at least 70%, but in the end the best we could acquire was at least 60%. The final result from running our trained model on the test data yielded an accuracy of 0.6094 or 60.94%. What would have helped us out a bit more would be to gain a deeper understanding of how the functions in our CNN architecture work.

Below is what our CNN architecture segment looks like:

```
FaceMaskClassification {
  (network): Sequential {
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU()
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU()
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU()
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU()
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (15): Flatten()
    (16): Linear(in_features=82944, out_features=1024, bias=True)
    (17): ReLU()
    (18): Linear(in_features=1024, out_features=512, bias=True)
    (19): ReLU()
    (20): Linear(in_features=512, out_features=6, bias=True)
  }
}
```

Evaluation

For the evaluation, we currently use a set of 400 images. First, we store the labels of each image in a list, and do the same thing for the neural network's predictions. This will yield two lists of integers (numbering 0 - 3 to represent each class), both of length 400. Cloth mask, N-95 mask, No Mask, and Surgical Mask being 0, 1, 2, and 3 respectively.

These lists are then used in Scikit-learn's “confusion_matrix()” function to create a multi-class confusion matrix. The highest values in the confusion matrix are visible along the diagonal, which you can somewhat tell what the accuracy of the neural network will be.

The appropriate formulae as well as Scikit's functions are used to calculate the accuracy, recall, precision and f1 measure for each of the four classes, as well as for the combination of all the classes.

So for the calculations of these metrics, the theoretical formulae from the lecture slides of manipulating confusion matrix values were used. However, it was also interesting to see what the results were for the total of the four classes. For those, sk-learn’s functions were used and gave the same values for all four metrics, which makes sense, given that False Positive and False Negative values were the same.

Multi-Class Confusion matrix:

```
****************************************
        Confusion Matrix
****************************************
Columns are predictions, rows are labels

[[60 14  7 15]
 [21 52  1 26]
 [15  2 85  4]
 [26 23  4 45]]
```

Metrics:

```
CLASS # 0
Accuracy:  0.76
Precision:  0.49
Recall:  0.62
F1_score:  0.55

CLASS # 1
Accuracy:  0.78
Precision:  0.57
Recall:  0.52
F1_score:  0.54

CLASS # 2
Accuracy:  0.92
Precision:  0.88
Recall:  0.80
F1_score:  0.84

CLASS # 3
Accuracy:  0.76
Precision:  0.50
Recall:  0.46
F1_score:  0.48


Total (all classes together):
Accuracy:  0.605
Precision:  0.605
Recall:  0.605
F1_score:  0.605
```

Cloth Mask	Predicted: Yes	Predicted: No
Actual: Yes	60	36
Actual: No	62	242

N-95 Mask	Predicted: Yes	Predicted: No
Actual: Yes	52	48
Actual: No	39	261

No Mask	Predicted: Yes	Predicted: No
Actual: Yes	85	21
Actual: No	12	282

Surgical Mask	Predicted: Yes	Predicted: No
Actual: Yes	45	53
Actual: No	45	257

References

[1] A. Jangra, “Face mask detection ~12K images dataset,” *Kaggle*, 26-May-2020. [Online]. Available: https://www.kaggle.com/datasets/ashishjangra27/face-mask-12k-images-dataset. [Accessed: 04-Jun-2022].

[2] D. Makwana, “Face mask classification,” *Kaggle*, 25-Jul-2020. [Online]. Available: https://www.kaggle.com/datasets/dhruvmak/face-mask-detection. [Accessed: 04-Jun-2022].

[3] W. Intelligence, “Face mask detection dataset,” *Kaggle*, 14-Jun-2020. [Online]. Available: https://www.kaggle.com/datasets/wobotintelligence/face-mask-detection-dataset. [Accessed: 07-Jun-2022].

[4] coffee124, “N95 face mask,” *Kaggle*, 05-Jun-2022. [Online]. Available: https://www.kaggle.com/datasets/coffee124/facemaskn95. [Accessed: 07-Jun-2022].

[5] P. Sharma, “Convolutional Neural Network Pytorch: CNN using pytorch,” *Analytics Vidhya*, 10-May-2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/. [Accessed: 04-Jun-2022].

[6] P. Soni, “Train CNN model with pytorch,” *Medium*, 25-Apr-2022. [Online]. Available: https://medium.com/thecyphy/train-cnn-model-with-pytorch-21dafb918f48. [Accessed: 04-Jun-2022].