Detector of Proper Face Mask

50.038 Computational Data Science

Final Report

Marcus Ho, Chung Zhi You, Mathias Koh

GitHub Repository Link:

https://github.com/marcushojww/properMaskWearingDetector

Dataset and Collection

MaskedFace-Net

For this project, we used a dataset consisting of human faces with a correctly or incorrectly worn face mask. We obtained this dataset from MaskedFace-Net (A. Cabani, 2020) which was based on the dataset Flickr-Faces-HQ (FFHQ).

Starting with a dataset of unmasked faces, they used computer vision to achieve facial detection and facial landmark detection in order to position masks onto the faces to create this dataset of correctly masked and incorrectly masked faces. The dataset contains a large variety in terms of age, ethnicity, viewpoint, lighting and image background; thus, it should result in a model that is able to work on and predict in more conditions.

Figure 1. Images from MaskedFace-Net dataset with proper mask usage, improper mask usage and no mask respectively

Prajna Bhandary's Dataset

We also used another dataset created by Prajna Bhandary (B. Prajna, 2020). She also started with a dataset of normal images of people's faces then used computer vision to detect their faces and their facial landmarks and used those landmarks to add masks into the pictures.

However, this dataset had only two labels for its dataset, namely with mask and without mask. It did not contain images of people wearing the mask improperly. Nonetheless, we aimed to use this dataset primarily for testing and to evaluate the efficacy of our later trained model.

Figure 2. Images from dataset created by Prajna Bhandary with mask worn and no mask respectively

Data Pre-processing

Data Extraction

The MaskedFace-Net dataset contained 133,783 images of human faces with a correctly or incorrectly worn mask, including images of human faces without a mask. Out of this huge pool available on OneDrive, we extracted about 700 images from each class and created a script named “check_dataset.py” which helps us identify the number of images we have in each class in our new dataset.

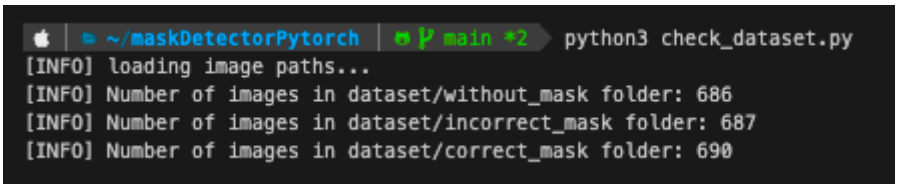

Figure 3. Number of images in each label present in our dataset

Train-Validation-Test Split

Once we were done with the data extraction, we proceeded to split our dataset into training, validation and test sets with a custom script. We first loaded all the image paths from our dataset folder and randomly shuffled them. This was needed to ensure that the images allocated to each set would be random.

With a train-test split of 0.2, we first split the dataset into 20% test and 80% train. Then with a train-validation split of 0.1, we further split the training dataset into 90% train and 10% validation. Finally, we have 1486 images in our train folder, 412 images in our test folder and 165 images in our validation folder (See Figure 4).

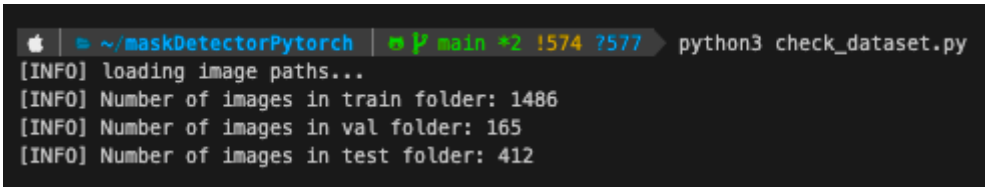

Figure 4. Number of images in train, test and validation folder

PyTorch ImageFolder and Dataloader

With PyTorch's ImageFolder class, it removes the hassle of manually arranging input data and its corresponding label in pairs, and allows us to store our image samples with their respective labels automatically. We can initialize the class by simply adding our root folder into the root argument of the class. Then with PyTorch's Dataloader class, it wraps an iterable around the ImageFolder dataset to enable easy access to our image samples.

```
# initialize the training and validation dataset
print("[INFO] loading the training, validation and test dataset...")
trainDataset = ImageFolder(root=config.TRAIN,
    transform=trainTransforms)
valDataset = ImageFolder(root=config.VAL,
    transform=valTransforms)
testDataset = ImageFolder(root=config.TEST,
    transform=testTransforms)
print("[INFO] training dataset contains {} samples...".format(
    len(trainDataset)))
print("[INFO] validation dataset contains {} samples...".format(
    len(valDataset)))
print("[INFO] test dataset contains {} samples...".format(
    len(testDataset)))

# create training and validation set dataloaders
print("[INFO] creating training and validation set dataloaders...")
trainDataLoader = DataLoader(trainDataset, batch_size=config.BATCH_SIZE, shuffle=True)
valDataLoader = DataLoader(valDataset, batch_size=config.BATCH_SIZE)
```

Figure 5. Loading our dataset with ImageFolder and Dataloader class

Data Augmentation

By creating fresh and varied images to our training datasets, data augmentation can help improve the performance and results of our CNN models. We believe our models will perform better and are more accurate when the dataset is rich and sufficient.

The Torchvision transforms library allowed us perform data augmentation through a series of alterations to the image such as the below:

1. transforms.RandomAffine
    a. Performs random affine transformation of the image keeping center invariant
2. transforms.Resize
    a. Resize the input image to the given size
3. transforms.RandomHorizontalFlip
    a. Horizontally flip the input image randomly based on the provided probability argument
4. transforms.RandomVerticalFlip
    a. Vertically flip the input image randomly based on the provided probability argument
5. transforms.RandomRotation
    a. Rotate the image based on the provided degrees argument

Implementation

For our data augmentation strategy, we decided to employ the image alterations in the order specified in the screenshot below (See Figure 6).

```
# initialize our data augmentation functions
resize = transforms.Resize(size=(config.INPUT_HEIGHT,
    config.INPUT_WIDTH))
hFlip = transforms.RandomHorizontalFlip(p=0.25)
vFlip = transforms.RandomVerticalFlip(p=0.25)
rotate = transforms.RandomRotation(degrees=15)

# initialize our training and validation set data augmentation
# pipeline
trainTransforms = transforms.Compose([resize, hFlip, vFlip, rotate,
    transforms.ToTensor()])
```

Figure 6. Initializing our data augmentation functions

With our data augmentation pipeline, we created a script to visualize the augmented images. We included transforms.RandomVerticalFlip to test the effect of vertically flipping the images in the train dataset to the model's performance.

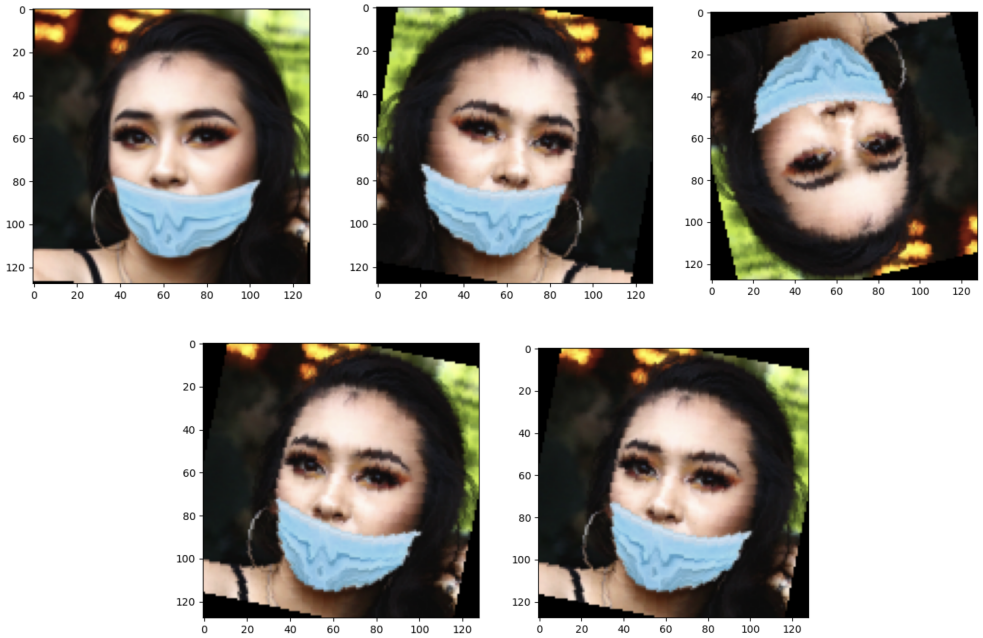

Figure 7. Result of transforming an image in our train dataset with our first augmentation strategy five separate times

Problem and Model

Problem

When we were assigned this project, we wanted to do something that was relevant to us not just in terms of what we were learning in this course, but also relevant to our lives. In the past two or so years, nothing has been more relevant to the Covid-19 pandemic and we wanted to do something that could help alleviate some of the problems brought about by this virus.

To combat the spread of the virus, one solution adopted all over the world is for the people to wear face masks, regardless of whether they have been infected or not, in order to mitigate virus transmission from person to person. This is especially important since studies have found that Covid-19 viral loads detected in asymptomatic patients were similar to those in asymptomatic patients (Zou et al., 2020). Unfortunately, the efficacy of the masks is heavily undermined by improper wearing of masks, especially among the elderly and the very young.

Currently, the solution to this problem is for human officials patrolling to see that a person is not wearing their mask properly and the official has to tell them to wear their mask properly. This is not an efficient solution as this is prone to human error and a human can only look at so much at one time. Thus, in an attempt to help come up with a solution to this issue of improper wearing of masks, we wanted to use computer vision and neural networks to help detect when someone is wearing a mask incorrectly so as to make the jobs of these officials easier or even be able to replace them with an automated system such that the manpower could be used somewhere more useful.

Convolutional Neural Network Implementations

Convolutional Neural Networks (CNN) was used as our model of choice as it serves as the standard form of neural network architecture for addressing image-related tasks. The characteristics of the CNN architecture such as local receptive fields, sub-sampling and weight sharing enable it to be a suitable candidate in tackling computer vision tasks.

1. LeNet Architecture

The LeNet CNN architecture was one of the earliest CNNs and was first proposed by Yann LeCun et al (*LeNet*, n.d.). It consists of 2 convolution layers, 2 max pooling layers and 2 fully connected layers (See Figure 8). The convolution layers allow us to extract key features from the input images while the max pooling layers reduce the dimensions of the feature maps, hence decreasing the number of parameters to learn and the required computation in the network. The output from the final pooling layer is then flattened and then passed into the fully connected layer. Finally, the fully connected layers serve as feed-forward neural networks and the final layer uses a log softmax activation function to get the probabilities of the input image being in a certain class.

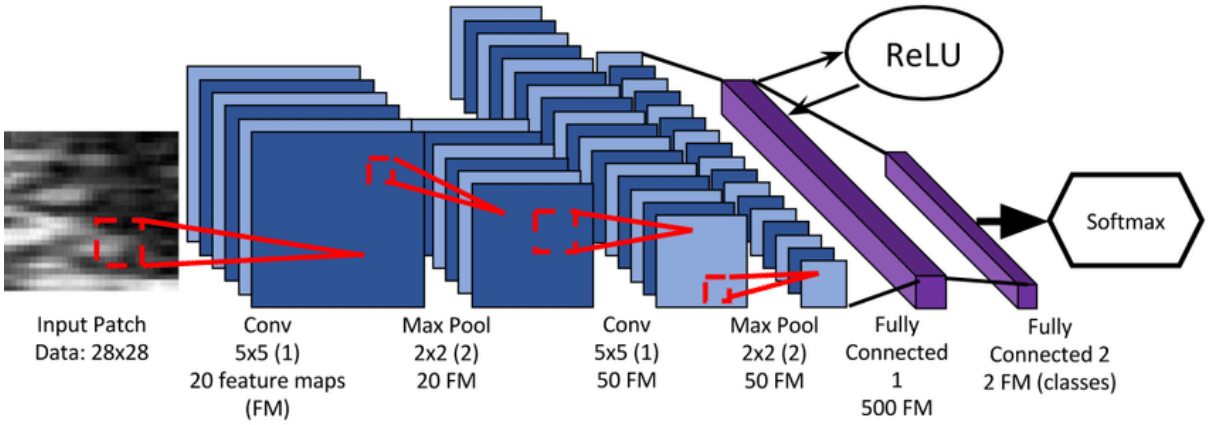

Figure 8. Visual representation of LeNet CNN architecture (Pons, n.d.)

We implemented the LeNet CNN architecture with a custom Python class which extends the torch.nn.Module class.

2. Resnet50 Architecture

ResNet50 is a variant of the ResNet model which allows for the network to include residual learning using shortcut connections (directly connecting input of the nth layer to some (n+x)th layer). The framework of ResNet made it possible to train ultra deep neural networks that could contain hundreds or thousands of layers yet still achieve good performance. ResNet50's architecture uses a 50 layer deep convolutional network consisting of 48. Convolution layers along with 1 MaxPool and 1 AveragePool layer.

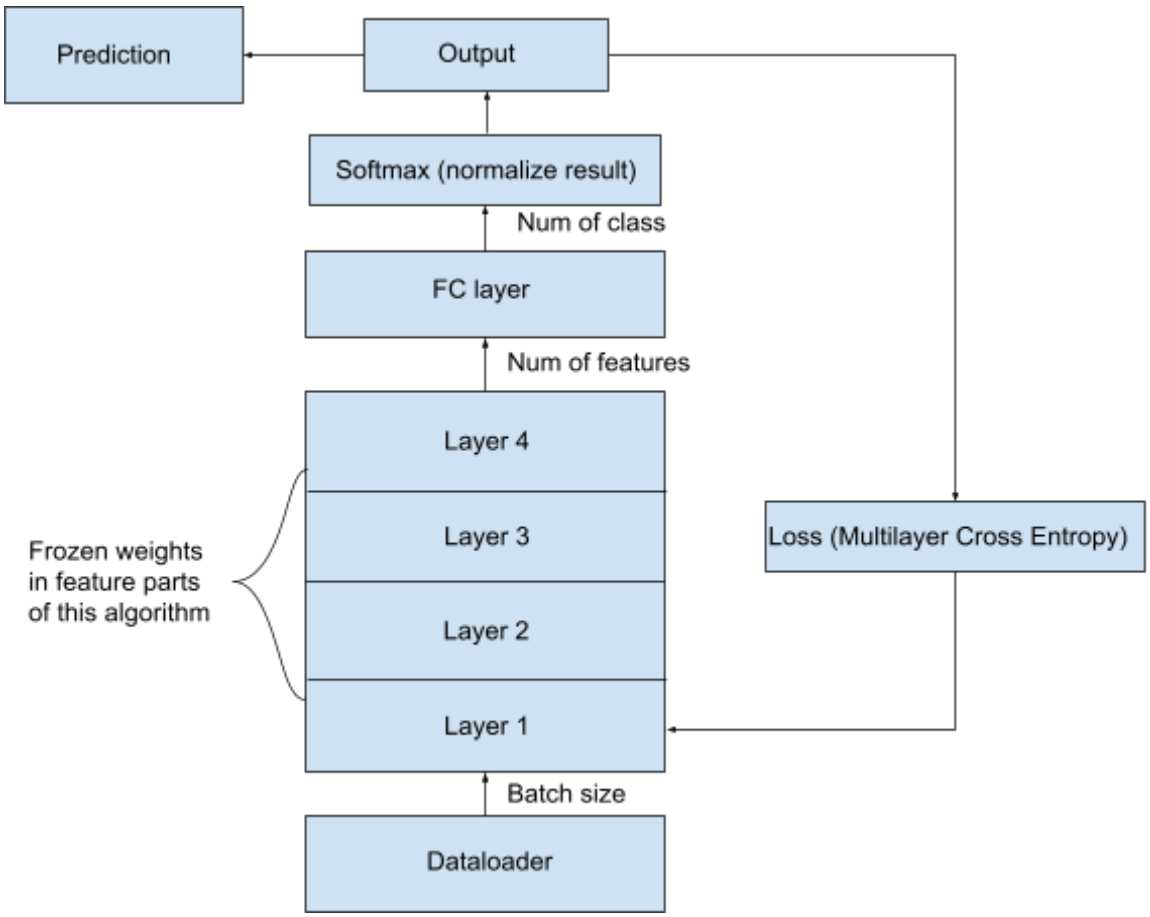

Figure 9. Visual representation of ResNet50 architecture

We made use of what we learnt in class such as transfer learning and feature extraction of the model and implemented in this project, by freezing the weights in feature training, we replace the fc layer with our own classification standards, first started by training the model for 100 epochs and study the loss and accuracy of both training and validation set, observing the global minimum and retrain until that epoch so to prevent overfitting of data.

3. GoogLeNet Architecture

GoogLeNet is a 22 layer deep convolutional neural network that is a variant of the Inception Network, a Deep Convolutional Neural Network developed by researchers at Google. The GoogLeNet architecture sought to solve the issue of large networks consisting of a large number of layers being prone to overfitting and suffering from either exploding or vanishing gradient problems. It does this with the Inception module's utilisation, leveraging feature detection at different scales through convolutions with different filters and reducing the computational cost of training an extensive network through dimensional reduction.

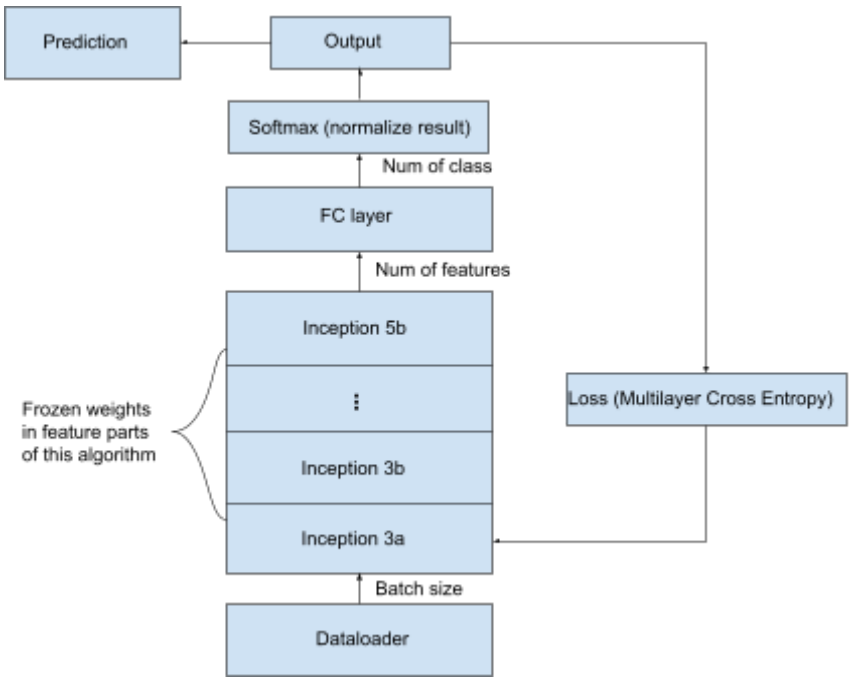

Figure 10. Visual representation of GoogLeNet architecture

GoogLeNet is a 22 layer deep convolutional neural network that is a variant of the Inception Network, a Deep Convolutional Neural Network developed by researchers at Google. The GoogLeNet architecture sought to solve the issue of large networks consisting of a large number of layers being prone to overfitting and suffering from either exploding or vanishing gradient problems. It does this with the Inception module's utilisation, leveraging feature detection at different scales through convolutions with different filters and reducing the computational cost of training an extensive network through dimensional reduction.

We used softmax as the activation function and MCE as loss function for our model, as we have to predict among 3 classes, and then by using argmax(1) to choose the class with the highest confidence score.

Evaluation Methodology

Loss Function and Optimizer

We implemented multi-class cross-entropy as an error function for our softmax activation function, which is a commonly used loss function for multi-classification tasks where our target is one-hot encoded vector.

For our optimizer, we implemented Adam, an adaptive learning rate optimization algorithm, which serves as a replacement for the stochastic gradient descent. Its algorithm can handle sparse gradients on problems that produce a lot of noise.

View Training and Validation Loss & Accuracy at Every Epoch

By calculating the train and validation loss and accuracy at every epoch, we can see if the model is potentially overfitting the training data. By calling torch.no_grad() which switches off autograd for evaluation, we can deduce that our model is overfitting if validation loss is high but that of the training set is low and continues to decrease.

Sklearn's Classification Report
After training the various models using the training dataset, we used the model to predict the labels of the test dataset and then used sklearn's classification report to measure the quality of the predictions.

The classification report used the number of true positives, false positives, true negatives and false negatives to predict the precision, recall, f1-score, support and accuracy. The precision is the percentage of positive predictions that were correct, the recall is the percentage of positive cases that were predicted correctly, f1-score is the weighted harmonic mean of precision and recall, support is the number of occurrences of that class in the dataset and accuracy is the percentage of total samples that were predicted correctly.

Between f1-score and accuracy, if the model is balanced (support of all the classes are similar), we can rely on accuracy for evaluating the model. However, if the model is unbalanced, the accuracy would be inaccurate and we would need to rely on the f1-score instead to evaluate the model.

Testing Our Model with Prajna Bhandary's Dataset

As we utilized the MaskedFace-Net dataset for our training, validation and test sets, we decided to do a final evaluation of our trained model's performance by testing it with Prajna Bhandary's dataset.

We wanted to determine if our model is only predicting well on the MaskedFace-Net dataset, and is unable to make accurate predictions on the other datasets which we did not train it on. This would also allow us to determine if the key features extracted from the CNN are good for prediction of any other images in the same class.

Comparison to Existing Work

We looked at one of the projects done by Adrian Rosebrock for his mask detection model on the pyImageSearch website. He applied transfer learning in MobileNet using pretrained weights from ImageNet, achieving around 99% in accuracy as observed from sklearn's classification report.

However one of the differences is that he predicts in 2 classes only, with mask and without mask, which we believe is easier in mask detection, where improper mask is the bigger challenge introduced when we just started including this new class. As it forces the model to recognise the face feature and the position of the mask in order before predicting.

Results and Discussion

Lenet Architecture

After training the model for 20 epochs, we observed the following classification report as we evaluated the model with the test set.

It can be seen that our model performs well in predicting the classes of the images in the test set (See Figure 11). Also, during the epoch runs, it can be seen that the training and validation loss approaches 0 while the accuracy approaches 1. As the validation loss and accuracy was improving with every epoch, we can infer that our model was not overfitting to our training data.

```
[INFO] total time taken to train the model: 757.73s
[INFO] evaluating network...
                precision    recall  f1-score   support

incorrect_mask       0.98      0.98      0.98       113
     with_mask       0.97      0.99      0.98       148
  without_mask       0.98      0.96      0.97       151

      accuracy                           0.98       412
     macro avg       0.98      0.98      0.98       412
  weighted avg       0.98      0.98      0.98       412
```

Figure 11. Classification report for our LeNet model through evaluation with test set

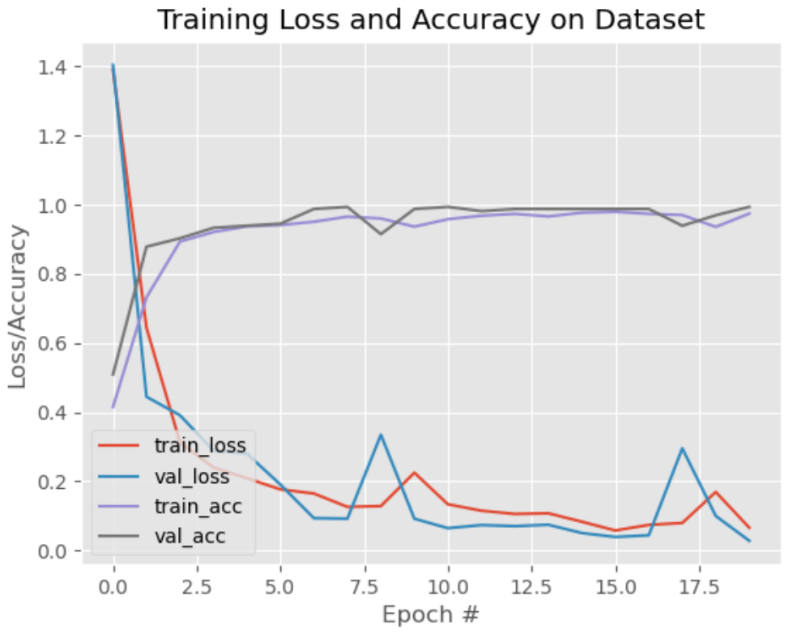

Figure 12. Training/Validation loss and accuracy graph for our LeNet model

To follow up with our evaluation methodology, we evaluated our LeNet model with Prajna Bhandary's Dataset. It is to be noted that Prajna Bhandary's dataset consists of two classes

only, namely “with_mask” and “without_mask”. Hence, we will focus on the f1-score for these two classes only from the resulting classification report.

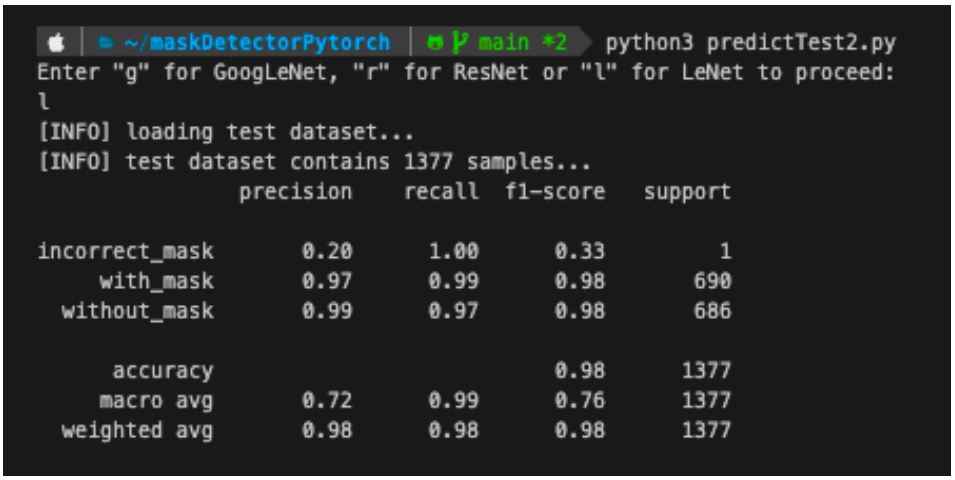

Figure 13. Classification report for our LeNet model through evaluation with Prajna Bhandary’s dataset

Our model performs well in predicting the classes from the external image dataset as seen from the f1-score of 0.98 for both “with_mask” and “without_mask” classes.

GoogLeNet

As for the pre-trained GoogLeNet model, we received the following results:

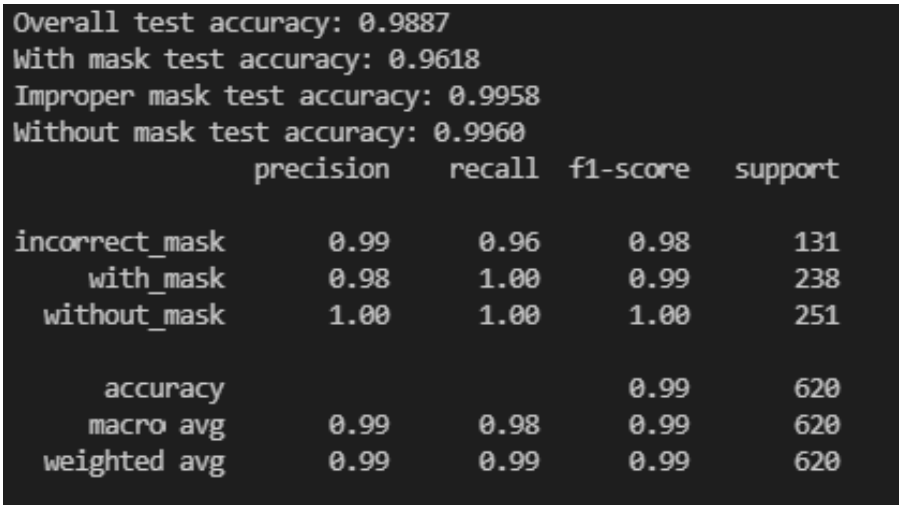

Figure 14. Classification report for GoogLeNet model through evaluation with test set

As we can see from the classification report, the f1-scores are very high for all 3 classes of prediction, with incorrect_mask slightly lower than the other two classes as we are guessing that incorrect mask may be a gray border between with mask and without mask.

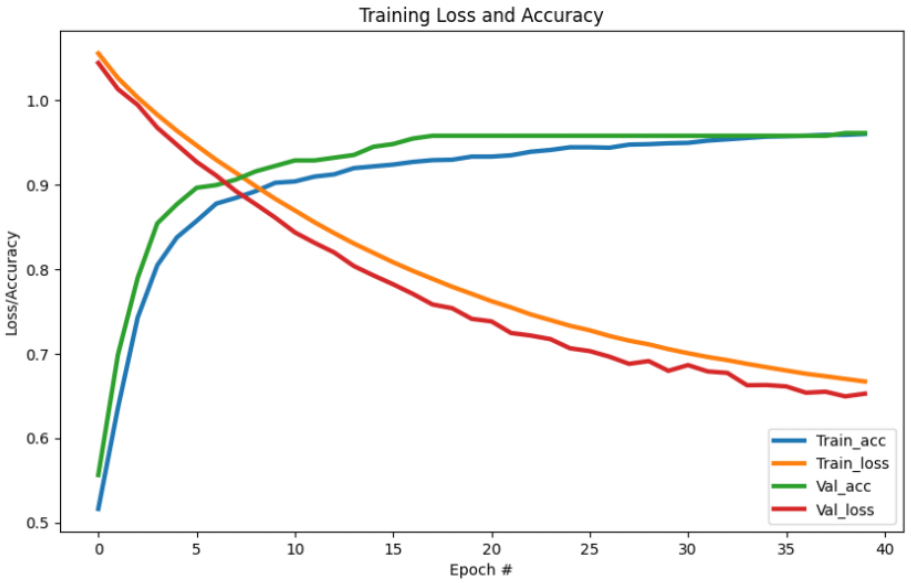

Figure 15. Training/Validation loss and accuracy graph for our GoogLeNet model

Another observation is that both the training loss and validation loss converged to around 0.6 after several epochs of training showing that we've achieved the state where further training will no longer improve the model much .

As the results are fairly good, our models are able to generalize well with new datasets feeding in. However, we have attached some images that the GoogLeNet model failed to predict with some reasoning behind why they might have failed:

1st Failed Prediction (Correct class: With mask, Predicted class: Incorrect mask)

```
chung@DESKTOP-SSAOITH:~/properMaskWearingDetector$ python3 GoogLeNetPredict.py
tensor([[0.7322, 0.1937, 0.0741]])
This image is improper mask.
With probability of 0.7321539521217346
```

With a probability score of 73%, we are guessing that it is very likely with the hand blocking parts of the mask giving the model the illusion of wearing the mask improperly. The classes are in the order of [improper mask, with mask, without mask], therefore after softmax it shows the 2nd highest of 19% confidence score in predicting the person is wearing a mask.

2nd Failed Prediction (Correct class: With mask, Predicted class: Without mask)

```
chung@DESKTOP-SSAOITH:~/properMaskWearingDetector$ python3 GoogLeNetPredict.py
tensor([[0.1492, 0.1573, 0.6935]])
This image is without mask.
With probability of 0.6934720873832703
```

With the confidence score of nearly 70% in predicting without mask we are guessing it may be due to the area of the face being too small as compared to the ratio in training sets, the model may be mistaken for the whole upper body as the face which we indicated by the circle, we can see both *improper mask* and *with mask* are of same confidence score due to this.

As for the evaluation of Prajna Bhandary's dataset, we observed the following:

```
[INFO] loading test dataset...
[INFO] test dataset contains 1377 samples...
               precision    recall  f1-score   support

incorrect_mask       0.04      1.00      0.08         1
     with_mask       1.00      0.93      0.97       690
  without_mask       0.97      1.00      0.98       686

      accuracy                           0.97      1377
     macro avg       0.67      0.98      0.68      1377
  weighted avg       0.98      0.97      0.97      1377
```

Figure 16. Classification report for GoogLeNet model through evaluation with Prajna Bhandary's dataset

As mentioned earlier in the evaluation methodology, we used our trained model to test on a new dataset to make sure that we did not not overfit on the current dataset and are generalizing well to predict any data. As the new dataset we are evaluating does not consist of incorrect_mask data, we insert 1 image to fulfill the 3 classes of prediction and prediction with mask and without mask only.

ResNet50

As for ResNet50, the general observation is quite similar to the GoogLeNet model with a few differences such as the training loss and validation loss has a slightly wider gap, this may be due to ResNet features in avoiding vanishing gradients with shortcut connections implemented so it takes longer to converge.

```
Overall test accuracy: 0.9919
With mask test accuracy: 0.9924
Improper mask test accuracy: 0.9958
Without mask test accuracy: 0.9880
                precision    recall  f1-score   support

incorrect_mask       0.97      0.99      0.98       131
     with_mask       1.00      1.00      1.00       238
  without_mask       1.00      0.99      0.99       251

      accuracy                           0.99       620
     macro avg       0.99      0.99      0.99       620
  weighted avg       0.99      0.99      0.99       620
```

Figure 17. Classification report for ResNet50 model through evaluation with test set

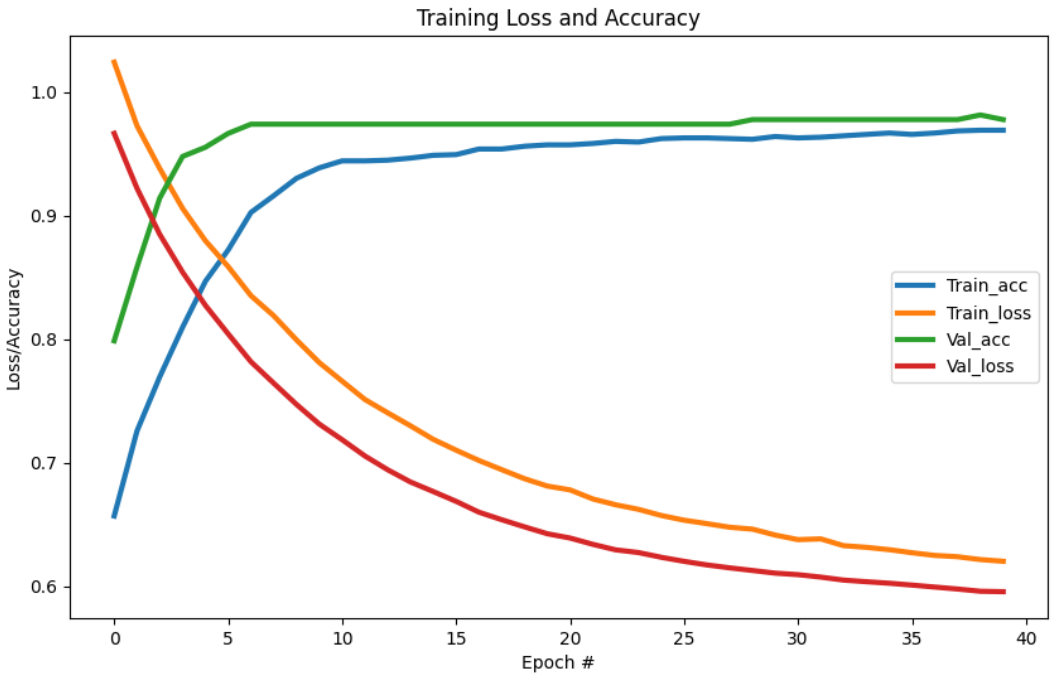

Figure 18. Training/Validation loss and accuracy graph for our ResNet model

As for the evaluation of Prajna Bhandary's dataset, we observed the following:

```
chung@DESKTOP-SSAOITH:~/properMaskWearingDetector$ python3 PredictDataset2.py
[INFO] loading test dataset...
[INFO] test dataset contains 1377 samples...
              precision    recall  f1-score   support

incorrect_mask       0.00      0.00      0.00         1
     with_mask       1.00      0.98      0.99       690
  without_mask       0.99      0.97      0.98       686

      accuracy                           0.98      1377
     macro avg       0.66      0.65      0.66      1377
  weighted avg       0.99      0.98      0.99      1377
```

Figure 19. Classification report for ResNet model through evaluation with Prajna Bhandary's dataset

Concluding from observations above, our model is still susceptible to a small percentage of inaccuracies where it could not detect certain images such as obstacles in front of the object and smaller than usual objects, however it is able to generalize well with most of the images fed in.

Summary and Future Improvements

With respect to the 3 models we implemented, LeNet, GoogLeNet and ResNet, we observe a common trend where all 3 models have an accuracy of 0.98 and above for their respective classification report with the test set. All 3 models also performed well in predicting the class from an external dataset, Prajna Bhandary's dataset, with an f1-score of 0.97 and above for the two classes "with_mask" and "without_mask".

We initially expected LeNet to perform significantly worse than GoogLeNet and ResNet. This is because our implemented LeNet architecture is only 4 layers deep as compared to ResNet which consists of deep networks of over 50 layers and GoogLeNet which is 22 layers deep. However, our findings show that LeNet can perform on the same level with ResNet and GoogLeNet for the chosen computer vision task, which is to detect proper mask wearing.

For future improvements, we could evaluate our models with more non-artificial datasets of images with the same 3 classes. Currently, the "incorrect_mask" and "with_mask" images in our dataset are created by adding face masks onto different parts of the face through the use of facial landmarks. Hence, we could observe if our models will be as effective in predicting real images and not only good at predicting artificially created ones.

References

Cabani, A., Hammoudi, K., Benhabiles, H., & Melkemi, M. (n.d.). *cabani/MaskedFace-Net: MaskedFace-Net is a dataset of human faces with a correctly and incorrectly worn mask based on the dataset Flickr-Faces-HQ (FFHQ).* GitHub. Retrieved December 5, 2021, from https://github.com/cabani/MaskedFace-Net

LeNet. (n.d.). Wikipedia. Retrieved December 2, 2021, from https://en.wikipedia.org/wiki/LeNet

Pons, G. (n.d.). *Fig. 3. The overall LeNet architecture. The numbers at the convolution...* ResearchGate. Retrieved December 2, 2021, from https://www.researchgate.net/figure/The-overall-LeNet-architecture-The-numbers-at-the-convolution-and-pooling-layers_fig2_318972455

prajnasb/observations. (n.d.). GitHub. Retrieved December 5, 2021, from https://github.com/prajnasb/observations/tree/master/experiements/data

SARS-CoV-2 Viral Load in Upper Respiratory Specimens of Infected Patients. (2020, February 19). NCBI. Retrieved December 5, 2021, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7121626/

Rosebrock, A. (2020, May 4). *COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning*. PyImageSearch. Retrieved December 5, 2021, from https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/