Face Mask Detector
Individual Report

Pragya Srivastava

Introduction

The goal of our project is to identify whether a person is **correctly** wearing a mask or not. A person correctly wears a mask when the mask completely covers his mouth and nose.

COVID-19, as we know, is a pandemic that has claimed millions of lives in the year 2020. Wearing a face mask has been identified as a successful method of preventing the spread of COVID amongst people. It is strongly recommended to wear a mask in public places. Most people follow the guidelines and wear masks. Some people do not wear it while others wear it incorrectly which doesn't cover their nose/mouth as it should.

Our project aims to train a model on images of people wearing masks and develop an interface to identify faces of people wearing the mask correctly, wearing it incorrectly or not wearing a mask.

For our project we used the MaskedFace-Net dataset[1]. This dataset is a synthetic dataset created using Generative Adversarial Network (GAN) on the Flickr-Faces-HQ Dataset[2]. The MaskedFace-Net model created synthetic images of people wearing the mask correctly and incorrectly. For our project we also wanted to identify whether the person was wearing a mask or not. So we added the original Flickr-Faces-HQ dataset images of people not wearing a mask to achieve this task.

The rest of the report will have the following structure - In the individual work section I will write about the work I did for the project and how it fits into the larger picture.

Individual Work - Background

For the purposes of this project we used Convolutional Neural Networks, in its standard form. My main task was to identify the model which would work best for this data and then finetune it. I explored pre-trained models - Resnet18, Resnet50, Alexnet, VGG, Densenet and Inception. I trained all of these models for 20 epochs with standard hyperparameters and then selected the one that gave me highest validation accuracy for further enhancements.

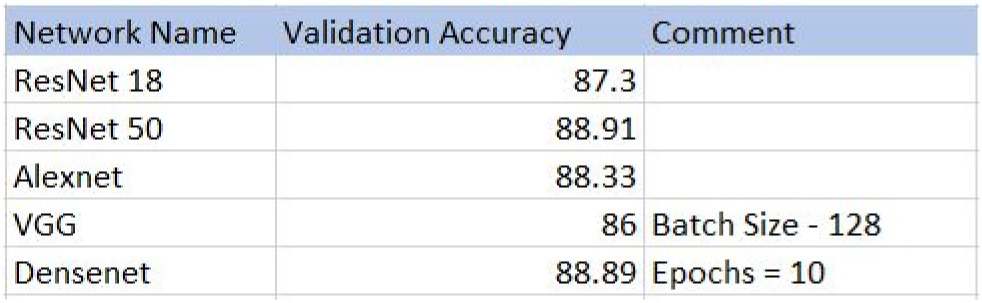
ResNet50 gave me the highest validation accuracy of about 88% without any tuning and hence was selected for this project. ResNet50 is a 50 layer Residual Network. ResNet, short for Residual Networks, is a classic neural network used for many computer vision tasks. ResNet trains very deep neural networks while overcoming the problem of vanishing gradients.

Once the ResNet was fine tuned I handed it over to the other team members so that they could use it for their model interpretability tasks.

Individual Work – Details

For this project, I used the data pre-processing scripts created by my team members. Once I had the data available on my AWS cloud, I started building the model.

I used the code available on the Pytorch documentation page to try out the various networks. The plan was to select the network which gave the highest validation accuracy and then fine tune it. I used the following networks, training the model for 20 epochs with a learning rate of 0.001 and batch size of 512. Some networks, like VGG were too complex and the batch size had to be reduced to 128 to run the model, otherwise it gave an 'out of memory' error. The results from the various models are given below –



Network Name	Validation Accuracy	Comment
ResNet 18	87.3	
ResNet 50	88.91	
Alexnet	88.33	
VGG	86	Batch Size - 128
Densenet	88.89	Epochs = 10

ResNet50 gave us the best results. The data was fed into the network with the following train, test and validation splits – 80:20 train and test, the validation was 20% of the train. The network was then fine tuned to increase accuracy. Since the dataset was balanced we did not need too many augmentations.

Transformations

Resize instead of RandomResizedCrop for train data – We used the pre-processing resize transform instead of the augmentation random resized crop because our data consisted of images which contain human faces in the entire frame in most cases, so our object of interest is within the frame.

ColorJitter Transformation – We used the ColorJitter transformation on the training data which randomly changes the color saturation of images to any percentage between zero and hundred. This helps in generalizing better masks of different colors.

We tried a few other augmentations like flipping and rotating images but those augmentations only worsened accuracy.

Hyperparameters

Learning Rate – Learning rate of 0.001 was used for this model. We tried 0.0001 as well but that worsened the accuracy metrics.

Batch Size – Batch size of 512 was used. In some cases if the cuda ran out of memory we reduced the batch size to 256 or 128 to run the model.

Optimizer – Adam was used as the optimizer.

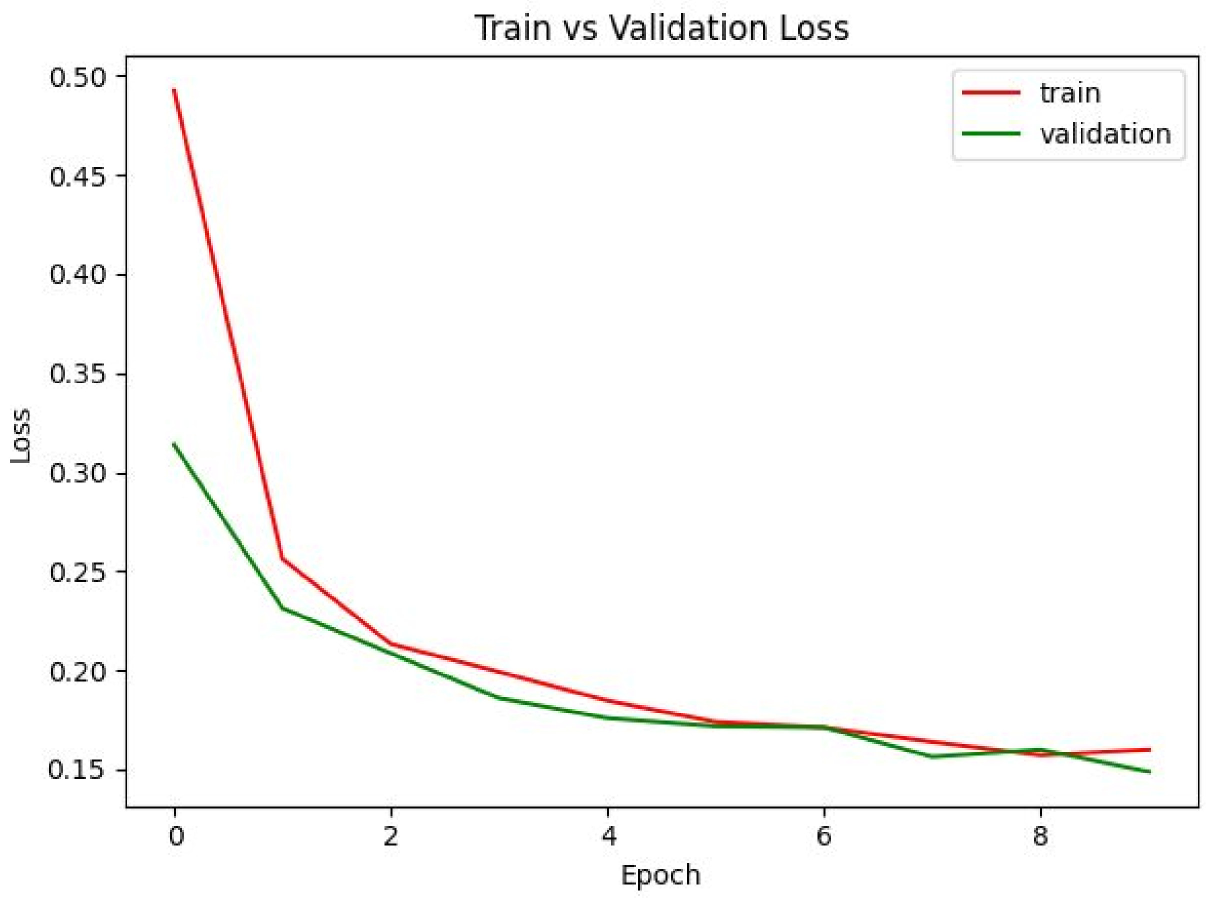
Epochs – The model was run for 10 epochs.

Regularization

Regularizing with Dropout – We have used 30% dropout rate in our model to prevent overfitting. We tried using 20% but 30% gives us the optimum results.

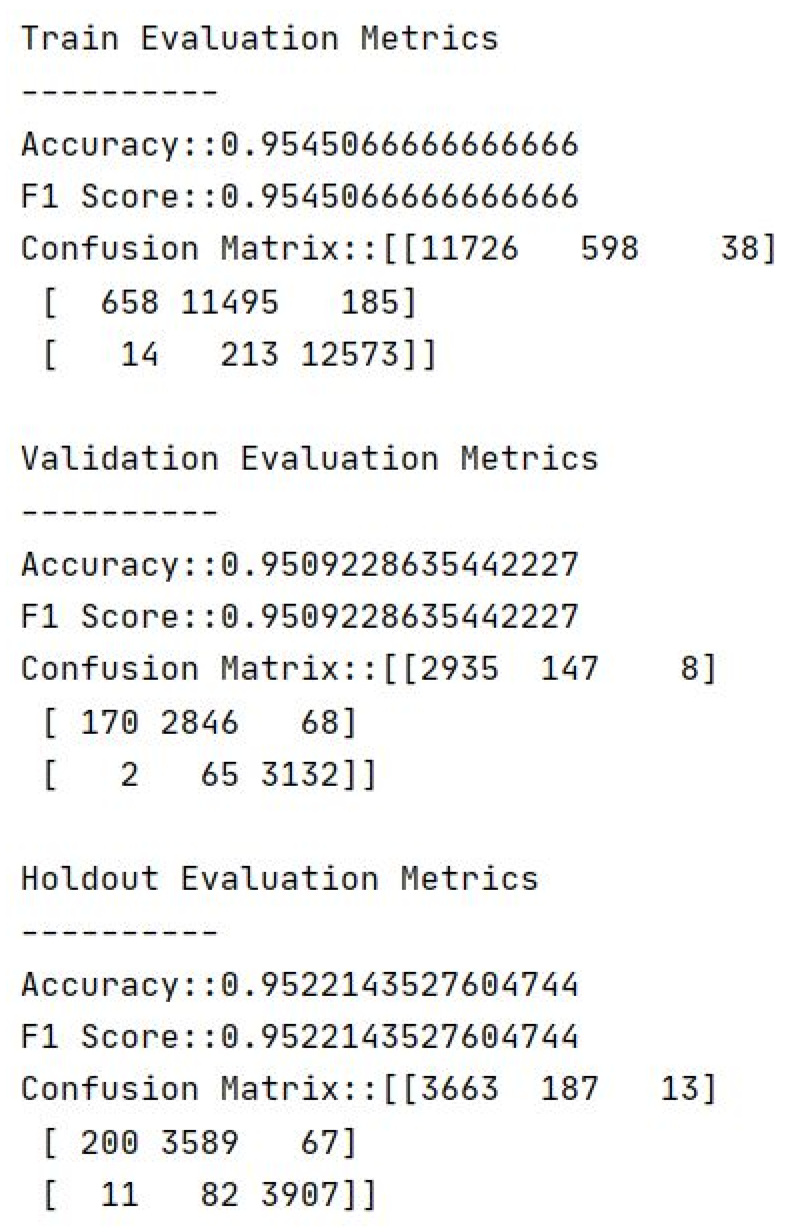
Model Accuracy

The performance of the model was judged based on accuracy and loss values for train, validation and holdout set. The loss on train and validation sets for ten epochs is given below –



The model does well since both train and validation loss are moving in the same direction for ten epochs and the validation loss decreases throughout the training.

Since the dataset was balanced, we have used accuracy as a metric to evaluate the model. The results for the final model are given below –



```
Train Evaluation Metrics
----------
Accuracy::0.9545066666666666
F1 Score::0.9545066666666666
Confusion Matrix::[[11726   598    38]
 [  658 11495   185]
 [   14   213 12573]]

Validation Evaluation Metrics
----------
Accuracy::0.9509228635442227
F1 Score::0.9509228635442227
Confusion Matrix::[[2935  147    8]
 [ 170 2846   68]
 [   2   65 3132]]

Holdout Evaluation Metrics
----------
Accuracy::0.9522143527604744
F1 Score::0.9522143527604744
Confusion Matrix::[[3663  187   13]
 [ 200 3589   67]
 [  11   82 3907]]
```

The final model has high validation and test accuracy. The F1 (micro) score equals the accuracy because the dataset was balanced.

Results

Once the model was trained with strong validation accuracy and low validation loss, other team members consumed the outputs for their experiments. My main learnings from this project exercise are as follows –

Architecture of other networks - Since I tried other networks, I learnt about their architecture, unique features etc. Some networks like Densnet perform well and some like VGG are extremely complex and need smaller batch sizes to work.

Transforms - I had previously not used color transforms on images, so using the ColorJitter transform on the train was something new that I learnt.

Hyperparameters - Training the model with various values for hyperparameters helped me get the best values for training the model, which gives the highest accuracy.

Summary and Conclusions

This project was extremely useful for me to understand how to use large datasets for training networks in the GPU. My contribution to the project included theoretical research to train large models using Pytorch. The steps that we followed to move the data to our AWS cloud location was something that I had never worked on, so it was a great learning experience.

Other than that this project gave me further practice on the Pytorch framework. Creating a pretrained model, and trying out different networks on the same data and evaluating performance was a learning experience. Computational scale of models with large datasets and computing tricks like parallelizing computation to ramp up training speeds was a tough task to do.

I really enjoyed working on this project and learned a lot about model interpretation and inference as well.

Code % - I wrote approximately 50% of the code and 50% was downloaded from the pytorch documentation pages and other web sources.

References

1. Kumar, N. (2019, December 17). Visualizing Convolution Neural Networks using Pytorch. Retrieved December 07, 2020, from https://towardsdatascience.com/visualizing-convolution-neural-networks-using-pytorch-3dfa8443e74e
2. https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html
3. https://towardsdatascience.com/different-types-of-regularization-on-neuronal-network-with-pytorch-a9d6faf4793e