Designing a Mask Classification Network Resistant to FGSM Attacks

Collin Warner, Alan Yu

May 20, 2021

1 Introduction

Over the past year the need to wear a face-mask has risen due the COVID-19 pandemic. Face-masks provide an effective way to slow the spread of COVID-19 with a majority of states requiring a face-mask to be worn at one point during the pandemic [6]. With this comes a need to determine whether a given person is wearing a face-mask or not. Stores could get an estimate of the number of people using masks and security could use it to make sure non-masked individuals are safely removed without endangering other customers.

To handle image classification, there has been a rise in popularity of using Convolution Neural Networks (CNNs)[4]. CNNs train with a sequence of layers starting with a convolution and ReLU activation layer, then a max-pooling layer. This sequence of layers can then be repeated. The final layers in a CNN are fully connected and lead to the eventual classification [12]. Training a robust network with a sufficient set of weights is time consuming, costly, and has led to the rise of transfer networks[2][1].

A transfer network is one where part of the network has been pretrained to solve a different problem and is reused in the current model. This reduces the overhead needed to train since a large portion of the weights have already been computed [1]. For this work we used VGG19 as our pretrained network because of its performance and since it was easily integratable in PyTorch [10]. We added two fully connected hidden layers and an output layer to scale the output down to our solution size and specifically train this model for classifying mask wearing. The first hidden layer used a tanh activation function and the second used a linear activation function. The output layer used a softmax activation function to get classification probabilities for the output layer.

In the extension of our work, we wanted to look at how to prevent network attacks that generate incorrect classifications. A common attack is the Fast Gradient Sign Attack (FGSM). This attack takes an image, and given access to the network, makes a bad image. The bad image is computed by taking the

original image $\mathbf{x}$ and target y, finding the gradient of the model on this image $\nabla_{\mathbf{X}} J(\theta, \mathbf{x}, y)$, then modifying the image to be $\mathbf{x} + \epsilon \cdot \text{sign}(\nabla_{\mathbf{X}} J(\theta, \mathbf{x}, y))$ for some small epsilon. This attack performs a gradient ascent, so it will maximize loss. [9]. This attack is powerful and important to prevent. With image classification, it could be possible to tamper with the images to make a large number of people look like they are wearing a mask when they are not. In the worst case, masks that exploit this vulnerability could be made and then worn incorrectly fooling cameras and rendering safety protocols ineffective.

To combat this, we added random Gaussian noise to training images on the belief that we wanted to be able to classify the same image even if it was slightly different than the original. We also used a model family approach. Similar to Uniform Hashing where a family of hash functions provide the necessary performance, we created a family of models that varied in weights but were trained from the same images. The intent was that an adversarial image for one model would not be an adversarial image for another, and so a classification with a random model from the family would have a decreasing chance of being effective the larger a family was used.

From preliminary results we found that training on images with Gaussian noise did provide a improvement to overall robustness from attack. We did not achieve effectiveness with a model family. Further discussion can be seen in Section 4.

In Section 2 we will go over the data we used, in Section 3 we detail our methodology, and in Section 4 we will summarize our results.

All code can be found at https://github.com/alany1/Mask_Classification_FGSM.

2 Data

The majority of our data was taken from the public data set Face Mask Detection on Kaggle [5].This dataset contained 853 images stored with an associated XML file. In each image, multiple people could be present. The XML file recorded the bounding box for where a face occurred and whether the face was one of 3 classes: wearing a mask, not wearing a mask, and wearing a mask incorrectly. We decided to ignore the 3rd class and only focus on creating a model that could classify the binary decision of wearing a mask or not. This decision was made because we had insufficient images of people improperly wearing masks. Without enough data, we were unable to train this section effectively and removed it to improve accuracy of the other two classes. After parsing the XML files, we cropped the image around each bounding box and made these crops into their own labeled images.

We then split up the data into folders containing only images with the clas-

sification type. In particular, folder 1 would contain all images of people with no mask on, while folder 2 would contain images of people with masks. This was done to make use of the torchvision.datasets.ImageFolder class.

The original dataset gave us around 3000 images of people with masks, but we were short on images of people without masks. Thus, our original model learned to predict masks with high probability, which led to a misleading test accuracy. To combat this, we added more images of people without masks to our data, from the Face Mask Lite Dataset on Kaggle [8].

3 Methods

3.1 Overview

To classify mask wearing, we used transfer learning by utilizing the pretrained VGG19 network from torchvision. The network consisted of three layers: features, average pool, and classification. The features and average pool components of our network were borrowed from the pretrained network. We modified the classifier to be small adding fully connected hidden and an output layers. There were two hidden layers that used a tanh and linear activation respectively. The output layer used softmax as its activation function. Dropout was also used with probability 0.25 on the first of these layers.

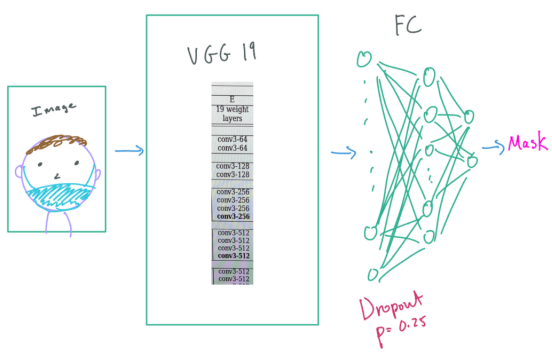


Figure 1: Sketch of the Network

3.2 A Closer Look at VGG19

VGG19 is one of many VGG (Visual Geometry Group) networks, consisting of 19 layers. It is an improvement of AlexNet, a CNN which achieved great performance during the ImageNet competition in 2012 [7].

The VGG19 network takes in a $224 \times 224 \times 3$ tensor as input to represent

an image. The input is then preprocessed by normalizing the tensor across each color channel to have mean $0.485, 0.456, 0.406$ and standard deviation $0.229, 0.224, 0.225$ for the red, green, and blue channels respectively. All kernels were of size 3×3 with stride 1 and spatial padding. All activations were ReLU. Below is a chart of several variants of the VGG model, with column E corresponding to VGG19 [3].

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2: Architecture of VGG model variants.

3.3 Simulating FGSM Attacks

Fast Gradient Sign Method (FGSM) attacks are one way of manipulating images to trick a neural network. It is known as a "white-box" attack because it requires knowledge of the network architecture.[9]

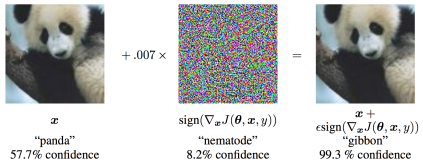


Figure 3: Basic idea of an FGSM Attack. [9]

For each image $\mathbf{x}$ and label y in the test set, we performed a gradient ascent on the image:

$$\text{new_image} = \text{old_image} + \epsilon \cdot \text{sign}(\nabla_x J(\theta, \mathbf{x}, y)),$$

which was then used as the new test image.

3.4 Training and Classification

As mentioned in Section 2, we had around 8000 training images. We budgeted 70 percent to training and the remaining 30 percent split evenly between vali-

dation and testing.

Our classification component consisted of two fully connected layers. The average pool layer from VGG19 gives 25088 inputs to layer 1, which then outputs 10 features. In this layer, we perform a dropout with 0.25 probability. These features are then passed into a tanh activation function and then into the second FC layer which outputs 2 features with a linear activation, into a 2 node output layer that used softmax for classification. When training, we only change the weights of this classification component and freeze the weights of the feature and average pool components from the pretrained VGG19.

Through experimentation, we found that a tanh activation worked better in our first classification layer. The reason behind this is unknown, though one possible reason is that tanh escapes the "dying relu" problem [11]. We also decided to make the classifier small. In the original VGG19 model classifier component, there were three layers with $\sim 123,633,664$ parameters to train but only a few thousand training images. Using a large network similar to this likely caused it to overfit, reducing its test performance. Thus, our smaller network with only $\sim 250,900$ parameters yielded better results.

For preprocessing, we included training, validation, and testing data transformations. In the case of validation and testing, we did a resizing and center crop to match the dimensions required for the VGG19 model. For training, we added a random 0 to 30 degree rotation followed by a random horizontal flip. We also added random Gaussian noise to each pixel of the image with mean 0.1 and standard deviation 0.08. We then trained the models for 6 epochs on a i7-8700K CPU and NVIDIA GeForce GTX 1070 Ti GPU.

We also attempted using a family of models, or training multiple models and for each prediction, selecting one at random with uniform probability and using it to classify the image. To create the adversarial image, we picked one model at random. To classify the adversarial image, we picked one again at random. The two processes were independent. This had a limited affect on accuracy and did not improve robustness to FSGM attacks, see Section 4.

Below is the training progression for one model across the 6 epochs and 10 validation tests.

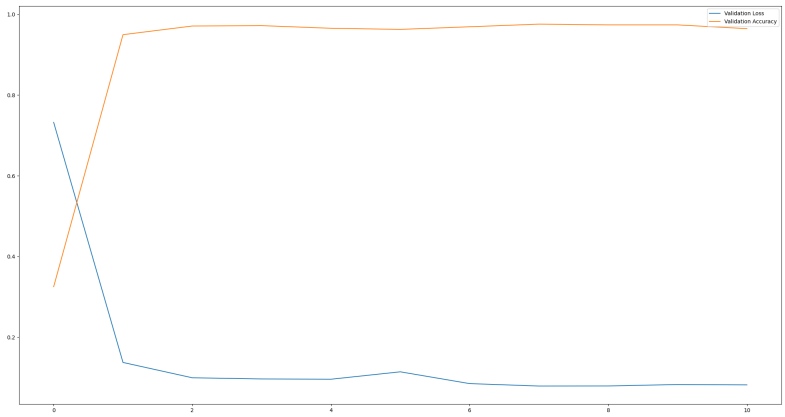


Figure 4: Training Progression

4 Results and Analysis

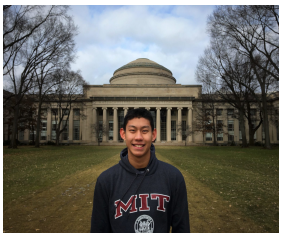
4.1 Mask Classification

The family model has a similar performance to a singular model. Without the family model, we get a 95.5 percent accuracy on the test set, while the family model gives an accuracy of 95.1 percent. These were both trained with Gaussian noise. However, when we do not include Gaussian noise during training, we reach an accuracy of 97.4 percent for a singular model. Below are some examples of classification:

Class: No Mask (0.995 confidence)

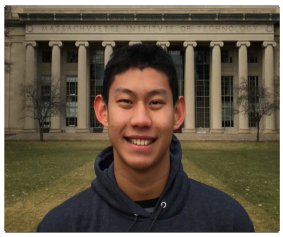
Class: Mask (0.997 confidence)

Although the network seems to be doing well at classifying, there are some rare cases where it fails. An explanation of this is given below.



Class: Mask (0.952 confidence)

Class: No Mask (0.997 confidence)



Class: No Mask (0.994 confidence)

As seen in the above images, the individual is not wearing a mask. However, the network fails in the first image and identifies him as wearing a mask. The third image is the exact same as the first image, only zoomed in. This trend is apparent across several images manually tested. We observe that the model is extremely accurate when the image is zoomed in on the individual's face, but when there is a non-trivial amount of background, the model has a good chance of failing. One way this can be improved on in future work is if we make predictions by first identifying a bounding box for the face and cropping, and then running the model on the cropped face.

4.2 FGSM Attacks

With an FGSM attack we are concerned with looking at the maximum difference from $\epsilon = 0$ which is when no attack occurs. The larger the difference, the more effective this attack was. We achieved the following results, which show that the maximum difference was 2.2% for the family model and Gaussian Noise.

Table 1: FGSM With Family Model and Gaussian Noise

Epsilon	Test Accuracy
0	0.874
0.2	0.852
0.4	0.859
0.6	0.863
0.8	0.859

With a single model, we got similar results, suggesting that the approach may not have been very significant. The maximum difference for this case was, 2.6%. One possible interpretation of this is that although the weights may vary between models, the gradients are still too similar and as a result there is no noticeable difference.

Table 2: FGSM With Singular Model and Gaussian Noise

Epsilon	Test Accuracy
0	0.877
0.2	0.851
0.4	0.857
0.6	0.862
0.8	0.861

Without adding Gaussian noise during training, the model breaks down under the FGSM attack, with a maximum difference of 20.6%. The fact that without Guassian noise our accuracy drops significantly enforces our hypothesis that training with Guassian noise allows the model to handle the slight deviations the attack makes on each image.

Table 3: FGSM Without Gaussian Noise

Epsilon	Test Accuracy
0	0.892
0.2	0.820
0.4	0.715
0.6	0.691
0.8	0.686

Although our data transformation was effective against FGSM attacks, there were some costs. Adding random noise slightly decreased our accuracy. However, we believe this problem could be addressed with more training samples reducing the impact of noise from a single image. Additionally we could train the model on the same image multiple times each epoch with different noise. This could lead to the model ignoring the noise and figuring out the true image. We also note that our approach of using a family model failed. Although we don't know the exact reason, the results seem to imply that an attack specific to one model that solves this particular classification problem is also effective against different, but similarly trained models.

5 Conclusion

Our model is quite accurate when classifying masks, but as mentioned in the Section 4, there are some pitfalls. Specifically, if the background of the image becomes too significant, then the model will fail. However, if the image is reasonably zoomed into an individual's face, the mask classification works with great accuracy. Additionally, we found adding Gaussian noise during training yields tremendous benefits when the model is subject to an FGSM attack. In our models, it seemed to be that this resilience is traded for accuracy, but we suspect that increasing the number of training examples and tuning hyperparameters

would be able to mitigate this problem. We also found that our approach of using a model family to prevent FSGM attacks was unsuccessful. This leads leads to further questions about how similar models are, that train to the same classification problem.

References

[1] J. Brownlee. A gentle introduction to transfer learning. URL https://machinelearningmastery.com/transfer-learning-for-deep\-learning/.

[2] B. Dickson. Why reducing the costs of training neural networks remains a challenge. URL https://bdtechtalks.com/2020/10/12/deep-learning-neural-networ\k-pruning/.

[3] A. Kaushik. Understanding the vgg19 architecture. URL https://iq.opengenus.org/vgg19-architecture/.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b\76c8436e924a68c45b-Paper.pdf.

[5] Larxel. Face mask detection dataset. URL https://www.kaggle.com/andrewmvd/face-mask-detection.

[6] N. M. Leah Asmelash and A. Watts. Texas ends its face mask rule on wednesday. URL https://www.cnn.com/2020/11/09/us/biden-mask-mandate-nationwid\e-trnd.

[7] NeuroHive. Alexnet – imagenet classification with deep convolutional neural networks. URL https://neurohive.io/en/popular-networks\/alexnet-imagenet-classification-with-deep-convolutional-neural\-networks/.

[8] prasoon kottarathil. Face mask lite dataset. URL https://www.kaggle.com/prasoonkottarathil/face-mask-lite-datas\et.

[9] Pytorch. Adversarial example generation. URL https://pytorch.org/tutorials/beginner/fgsm_tutorial.

[10] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[11] Stanford. Cs231n convolutional neural networks for visual recognition. URL https://cs231n.github.io/neural-networks-1/#actfun.

[12] M. Stewart. Simple introduction to convolutional neural networks. URL https://towardsdatascience.com/simple-introduction-to-convolut\ional-neural-networks-cdf8d3077bac#:~:text=There%20are%20three%\20types%20of,Features%20of%20a%20convolutional%20layer.