Implementation of Machine Learning Techniques on Detecting Face Mask

TingJun Chen
University of Oregon
1585 E 13th Ave, Eugene, OR
tchen3@uoregon.edu

YuShu Chen
University of Oregon
1585 E 13th Ave, Eugene, OR
yushuc@uoregon.edu

Abstract

In this paper, we constructed several machine learning techniques on detecting face mask wearing situation. Also, we compared and discussed the hyperparameter optimization in a customized convolutional neural network.

1. Introduction

Since the start of the pandemic, masks have become essential in preventing people from being infected from human activities. Most of the organization and public space require masks to be worn in order to let the person get into the building, especially in places such as the hospital, wearing a mask protects both doctors and clients from spreading the virus. However, checking whether a person is wearing a mask or not can be difficult and inefficient due to a large number of people entering and exiting the building. In the past, the only solution was to have staff at the entry to check, now; we want to implement the machine learning solution to the problem by creating a network to automatically check for mask wearing.

2. Problem Statement

Our goal is to train a convolutional neural network and other machine learning models, in order to classify the input images into class wearing mask or class not wearing mask.

3. Data Sets

Our dataset is acquired from the Kaggle website [1]. The training set contains 2500 images of a person with mask on and 2500 images of a person without mask. Similarly, the testing set contains 1000 images with mask on and 1000 images with mask off.

The size of each image varies, approximately around 150*100 and contains 3 RGB channel; during the training, however, the image will be resized to 256*256. Then, the total feature size is 3*256*256=196608.

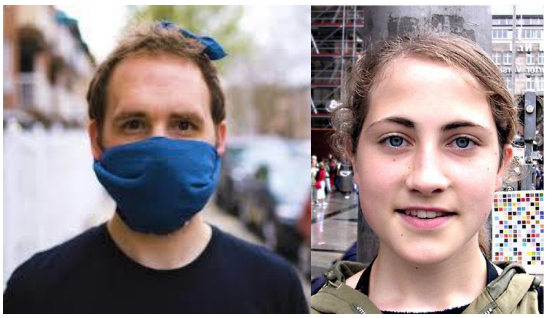


Figure 1: Sample of images of wearing mask (left) and not wearing mask (right).

4. Methods

4.1. Convolutional Neural Network

4.1.1 Custom Model Architecture

First, we implemented a custom convolutional neural network as a baseline model.

4.1.1.1 Convolutional

The convolutional layer acts as an extraction tool to extract the most important feature in the image; at the same time, shrinking down the size of the image and enhancing the special feature of the image.

Specifically, in the convolutional layer, the first layer is a two-dimensional convolution layer with input channel of 3, output channel of 20 and kernel size of 3; the second layer is a batch normalization layer, batch normalization is one of the regularization methods to recenter and rescale the input data.

$$\widehat{xi} = \frac{xi - \mu\beta}{\sqrt{\sigma^{2\beta} + \epsilon}} \quad (1)$$

Here, $\mu\beta$ is the mean of the batch and $\sigma^{2\beta}$ is the variance of the batch.

The third layer is an activation function layer with ReLU activation. ReLU function prevents the model from exponential scaling growth rate in the model computation time. [2]

The sequence of convolutional layers consists of multiple groups of these three layers, except the output channel size is increasing. In group 4, a stride size of 2 is introduced to reduce the size of the image. Similarly, in group 8 and 12, two more stride sizes of 2 were added. After the sequence of layers, the output feature size is 512*25*25, with a channel size of 512 and image size of 25*25.

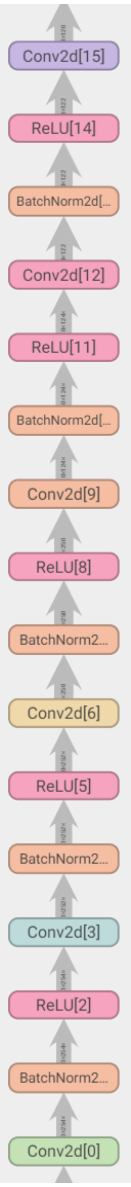


Figure 2: A section of the Convolutional Layers.

4.1.1.2 Image After Feature Extraction

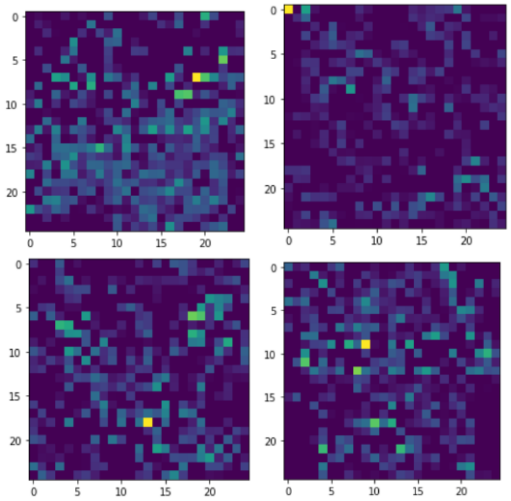


Figure 3: Here, we are able to identify the outline feature of a mask in the 1st, 3rd and 4th image.

4.1.1.3 Fully Connected

Followed by the convolutional layers are the fully connected layers. These layers match the input features with the output class, via a certain number of neurons representing an arbitrarily complex function.

The first layer takes the input size of 512*25*25 from the convolution and outputs it to 500 hidden neurons; the second layer is the ReLU activation function. In the third layer, we introduced the dropout layer that randomly drops a feature that has little importance in explaining the input, dropping out prevents the model from overfitting due to having useless features.

4.1.2 Algorithms

4.1.2.1 Stochastic Gradient Descent

The learning process involves the method of stochastic gradient descent. Mathematically, traveling in the direction of the gradient is the fastest way to travel from one level curve to another level curve because of the nature of the gradient that is perpendicular to the tangent plane at a certain point. Therefore, by stepping down in the direction of the gradient, the model is able to reduce the error and achieve some optimization.

$$\theta j = \theta j - \frac{\alpha}{m}\sum_{i=1}^{m}[(h\theta(x_i) - y_i)x_i] \quad (2)$$

Here $\frac{\alpha}{m}$ is the learning rate of the model.

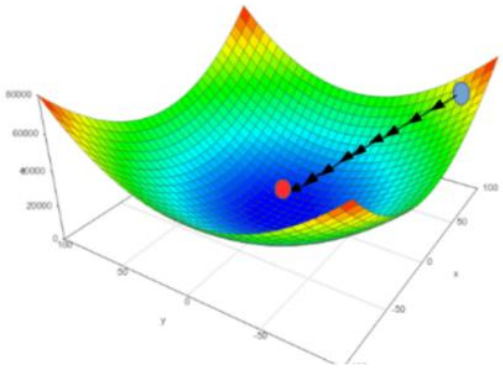


Figure 4 [3]: Stepping down the level curve.

4.1.2.2 Cross Entropy Loss

To evaluate the performance on training, cross entropy loss is used. The cross-entropy loss is a way to measure how much the two predicted probability distributions differ from the two actual probability distributions over a set of events.

$$L = \sum_{i=1}^{2} t_i * \log(p_i) \quad (3)$$
$$= -[tlog(p) + (1-t)\log(1-p)]$$

Here, t is the true probability and p is the predicted probability.

4.1.3 Transfer Learning with ResNet

One of the challenges in deep network training is that it's difficult to learn and update the weights because the gradient is approaching zero towards the end of the network. However, ResNet provides a solution to the problem by introducing the residual block, in which the network can learn from the residual output, rather than the direct output, unlocking the potential of the deep network.

We decided to implement transfer learning by fine tuning a pretrained ResNet; by doing so, we have a metric for comparison to the baseline model. In particular, most of the ResNet layers are maintained unchanged, except the output channel of the final fully connected layer is set to 2 to coordinate with the number of classes (masked vs unmasked).

4.1.4 Testing Result and Discussion

4.1.4.1 Hyperparameters

To decide the best hyperparameters, we are comparing the accuracy of different values on the following parameters.

4.1.4.1.1 LR Scheduler

Test Cases: No Scheduler vs. Scheduler "ReduceLROnPlateau"

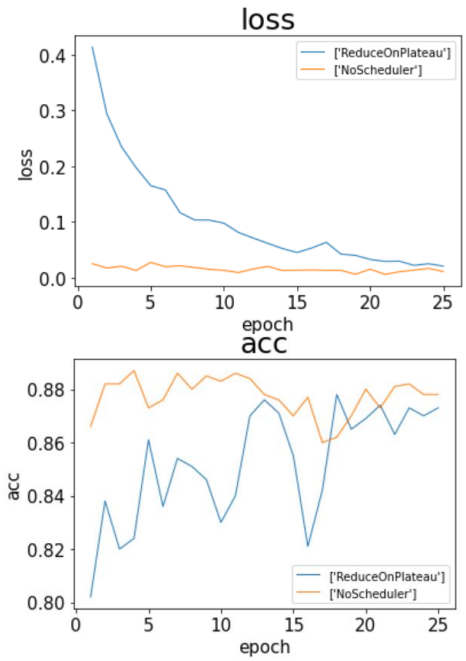


Figure 5

Here, we can see that around epoch 16, the loss of the scheduler model increased and the accuracy of the model decreased dramatically and bounced back up after that, potentially because of the change in the learning rate. Overall, the model with the default learning rate: 0.005 and stabilized at 0.05 has a higher accuracy than the model with scheduler.

4.1.4.1.2 Data Augmentation

To test the effect of the data augmentation, we tested two cases: one with the basic transformation: resized to 256 and to tensor; one with more transformation: resized, color jitter, rotation, horizontal flip, vertical flip, gray scale and to tensor.

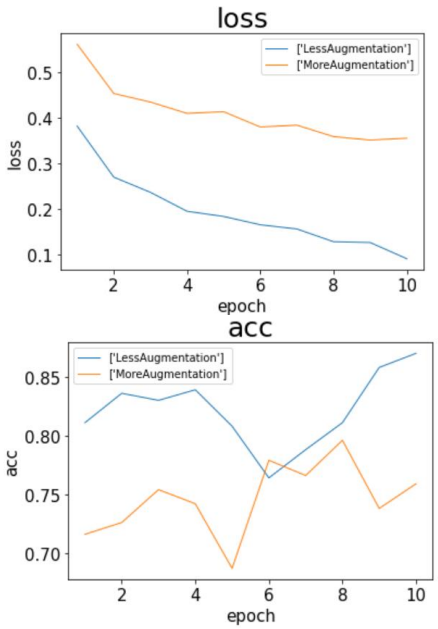


Figure 6

Here, we can see that the loss of the model with basic augmentation is constantly lower than the model with more augmentation by approximately 0.2, and the accuracy is higher by approximately 10%.

4.1.4.1.3 Weight Decay

Test Cases: No Weight Decay vs. 0.3 Weight Decay

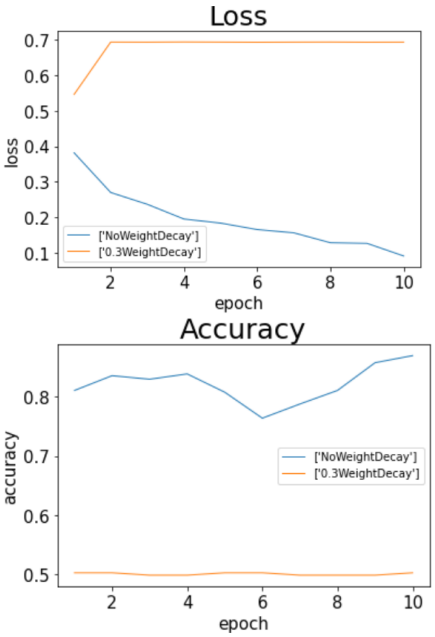


Figure 7

Here, we can see that the loss of the model with 0.3 weight decay has no trend of decreasing and stabilizing at 0.7, same with the accuracy, which stayed around 0.5.

4.1.4.1.4 Dropout Rate

Test Cases: No Dropout Rate vs. 0.3 Dropout Rate

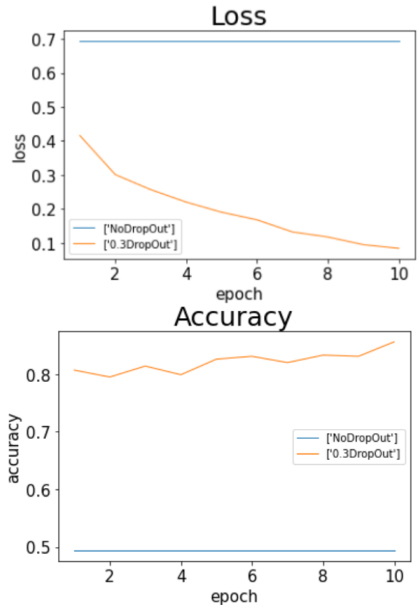
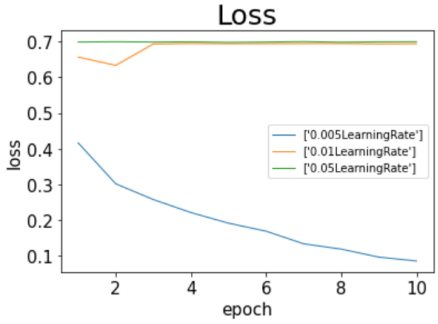


Figure 8

Here, we can see that dropout rate is one of the most important parameters in the model, without dropout, the loss doesn't have the trend of decreasing and the accuracy is not increasing because insignificant features are causing the model to perform poorly.

4.1.4.1.5 Learning Rate

Test Cases: 0.005 LR vs. 0.01 LR vs. 0.05 LR



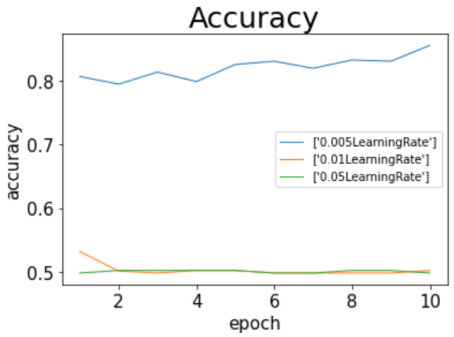
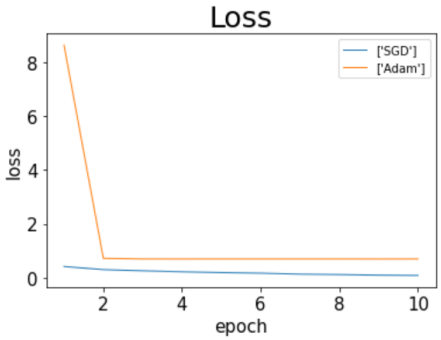


Figure 9

Here, we can see that the loss of the model with learning rate of 0.01 and 0.05 is not decreasing, potentially because the loss is jumping from one side of the loss surface to the other size of the loss surface and never stepping down ward, which is too large for the model.

4.1.4.1.6 Optimizer

Test Cases: SGD vs. Adam



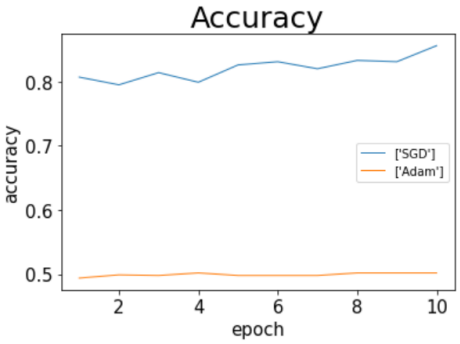


Figure 10

Here, we can see that the stochastic gradient descent optimizer performs much better than the Adam optimizer, while the accuracy of the Adam optimizer stays around 0.5.

4.1.4.1.7 Parallel Coordinate Plot

We used Weights & Biases sweep server to construct the parallel coordinate plot between hyperparameters.

See Figure 11.

4.1.4.2 Custom Model vs. ResNet

We tested the optimized custom model and ResNet101 with 20 epochs.

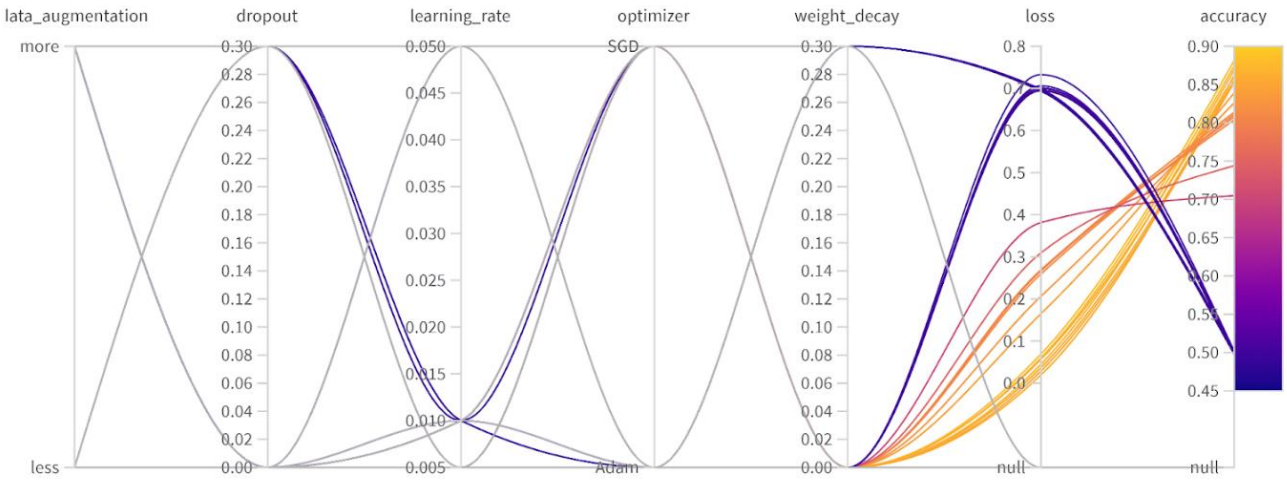


Figure 11

ResNet101:

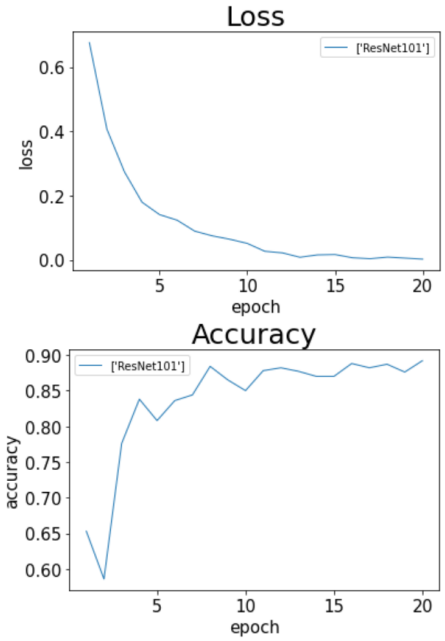


Figure 12

Custom Model:

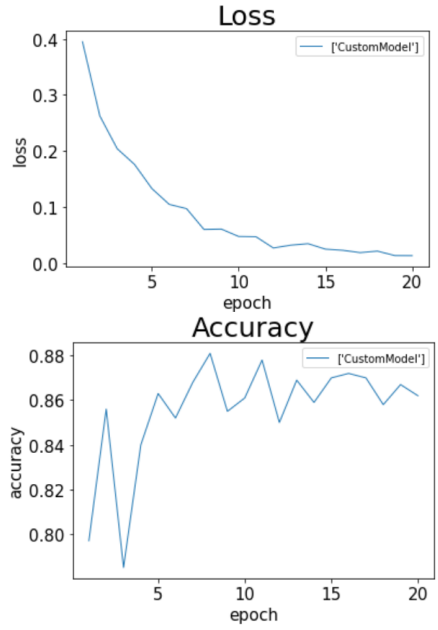


Figure 13

Here, we can see the loss curve is very similar while the accuracy of the RetNet is approximately 2% higher than the custom model. However, looking at the accuracy curve of the custom model, there was still a noticeable amount of fluctuation near the end of the epochs, potentially because the learning rate is fixed and we are not reaching the optimal number of epochs. The accuracy of the custom model still has the potential to be higher than current value. In term of the training time used, ResNet is faster, with 250 seconds per epoch for ResNet and 335 seconds per epoch for custom model. Overall, ResNet is a better choice.

4.2. Supported Vector Machine

Support Vector Machines are machines that are widely used to classify objects. In the SVM algorithm, we plot each data item as a point in n dimensional space, where the n is the number of features, we have, and the value of each feature is the value of a specific coordinate. We then perform classification by finding a hyperplane that distinguishes these two classes well.

$$\min_{\omega,b,\xi} \frac{1}{2w^T w} + c\sum_{i=1}^{n} \zeta i \tag{4}$$

Here, c is the regularization parameter. Additionally, from a point x to the Hyper Plane distance can be show as:

$$\gamma = \frac{|\vec{w}\cdot\vec{x}+b|}{\|\vec{w}\|} \tag{5}$$

Here, we want to maximize γ, which is to maximize $\frac{1}{\|\vec{w}\|}$, Then, we have the constraint:

$$y_i(\vec{w}\cdot\vec{x}_i+b) \geq 1, i = 1,2,\dots,m \tag{6}$$

or more generally:

$$y_i(w^T\phi(xi)+b) \geq 1-\zeta i \tag{7}$$

Before testing, we flatten the training set into a 1d array with 196608 features, then we used SVM function from scikit-learn to train the model. The accuracy is 0.844.

4.3. Naïve Bayes

Naïve Bayes models calculates the probability of a certain situation given the probability of another situation.
The equation is the following:

$$P(C_k|X_1,\dots,X_n) = P(C_k)\prod_{i=1}^{n}\frac{P(X_i|C_k)}{P(X_i)} \tag{8}$$

or more intuitively:

$$P(A/B) = P(B/A) * P(A)/P(B) \quad (9)$$

Here, the formula expresses the conditional probability of event A under the condition of event B, which is equal to the conditional probability of event B under the condition of event A multiplied by the probability of event A, divided by the probability of event B.
Similar to SVM, we first flatten the image data than train the model, with accuracy of 0.753.

4.4. Logistic Regression

Logistic regression is a machine learning method for binary classification (0 or 1) problems to estimate the likelihood of a certain event.
The equation is the following:

$$y = \sigma(f(\boldsymbol{x})) = \sigma(\boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w}^T\boldsymbol{x}}} \quad (10)$$

Here, W^T is a set of weights. As long as the X we need to predict is substituted into the above equation, the output y value is the probability of the label, and we can determine which category the input data belongs to. The accuracy is 0.808 after training with flattened data.

5. Conclusion

During the experiments, we gained several insights. Before adding Batch Normalization layers and Dropout layers, the model performs decently in the training set but poorly in the testing set; after the layers were added, accuracy of the testing set increased dramatically. Across all the models we tested, ResNet101 was the highest in accuracy, but we also saw the potential in the custom model if the learning rate was more optimized.

6. Contribution

TingJun Chen: Designing, training, testing CNN models, hyperparameters optimization, section 1-4.1 of the report.
YuShu Chen: Data visualization, other models, section 4.2-4.4 of the report.

7. Code

Our code is on GitHub:
https://github.com/theonering3/CIS-473-Machine-Learning

References

[1] "Face Mask Detection Dataset". Kaggle.Com, 2022, https://www.kaggle.com/omkargurav/face-mask-dataset. Accessed 11 Mar 2022.

[2] baeldung. “How ReLU and Dropout Layers Work in CNNs | Baeldung on Computer Science.” Baeldung on Computer Science, 30 May 2020, www.baeldung.com/cs/ml-relu-dropout-layers. Accessed 11 Mar. 2022.

[3] "Gradient Descent At Your Fingertips". Linkedin.Com, 2022, https://www.linkedin.com/pulse/gradient-descent-siddhant-mittal?articleId=6679410436036706305. Accessed 11 Mar 2022.