An Application of CNN: 3-class Face Mask Detection

Haoming Liu
hl3797@nyu.edu

Kaiwen Dai
kd1860@nyu.edu

Chunli Xu
cx523@nyu.edu

Abstract

This project aims to detect whether people are wearing their masks correctly or not using convolutional neural networks. Our intended outcome is to distinguish people who are correctly masked, incorrectly masked (for example, have their nose out), and unmasked. We tried two different ways to tackle this problem. We started by treating face mask detection as a two-staged problem: recognize the face and then classify the face into three categories. Also, We applied YOLO V4 to realize localization and classification at the same time. We found that the two-stage solution provided higher accuracy while performed relatively poor on face localization. On the other hand, YOLO V4 performed much better on localization but had a disadvantage in classifying incorrectly masked due to our data's limitation.

1. Introduction

Masks play a crucial role in protecting individuals' health against respiratory diseases, as is one of the few precautions available for COVID-19 in the absence of immunization. Although the covid-19 pandemic has been effectively contained in China, the prevention and control efforts should not be too lax, and wearing masks in public places is still essential. Checking whether people are wearing their masks correctly when there is heavy traffic is always difficult and requires much labour.

Many prior works have been working on utilizing machine learning to detect face masks and have achieved different results. For instance, AIZOOTech [1] on GitHub made a 2-class (with or without face mask) detection model using the structure of SSD. Their model consists of only 24 layers with the location and classification layers counted and 1.01 million parameters. They achieved a good PR curve with a great AUC (area under curve). They also deployed a web page that enables users to input real-time videos and detect multiple people wearing masks [2]. In the case of 3-class (with, without, and incorrectly wearing the mask) detection, for example, Shivam Saini [3] on Kaggle built a CNN model and used data augmentation and dropout to avoid overfitting. However, his final accuracy is no more than 0.8. We saw both outstanding achievement and large space for improvement.

Based on this, we decided to conduct our own project on 3-class mask detection. A practical application would be using this system to detect whether a student is correctly wearing his or her mask when entering the academic building. We hope that this project will contribute to the regular prevention and control of the pandemic by regulating people's behavior.

2. The Dataset and Features

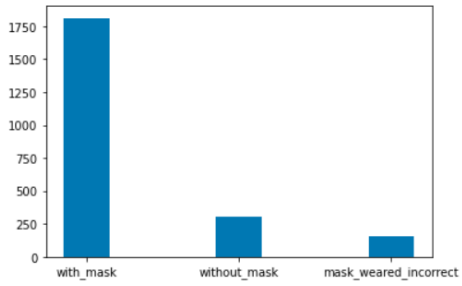
We utilized the dataset *Face Mask Detection* by Larxel on Kaggle [4]. This dataset contained 853 images with faces belonging to 3 classes, as well as their bounding boxes in the PASCAL VOC format. Though we found multiple datasets at the early stage of this project, we finally choose to dedicate to this one since

[1] https://github.com/AIZOOTech/FaceMaskDetection
[2] https://demo.aizoo.com/face-mask-detection.html
[3] https://www.kaggle.com/shivam541n1/face-mask-detection
[4] https://www.kaggle.com/andrewmvd/face-mask-detection, publication date: 2020/05/22

it is the only dataset containing the data of bounding boxes of faces. With this dataset, it is possible to create a model to detect people wearing masks, not wearing them, or wearing masks improperly. We also noticed that pictures in this dataset could contain more than one person. These images were also likely to have the corresponding annotation recorded more than one object. For instance, in a picture containing three people, two might be labeled as 'without_mask,' one might be labeled as 'with_mask.' To preprocess the raw dataset, we read the corresponding image's file path and all the information about the objects in the image from the XML files, including their label and the region of interest. We obtained in total 4072 faces, 717 of which belonged to 'without_mask,' 3232 belonged to 'with_mask,' and 123 belonged to 'mask_weared_incorrect.'

To deal with the imbalance of the data and improve the quality of input images, we filtered and removed faces whose size was smaller than 20, resized faces to 64×64, and applied horizontal flip to incorrectly masked faces. In the end, we had a dataset consisting of faces with 1811 labeled 'with_mask', 308 labeled 'without_mask', and 158 labeled 'mask_weared_incorrectly.'

Figure 1. Final Data Distribution



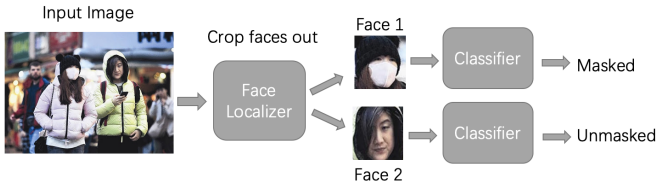
From Figure 1, we can still observe the imbalance in our dataset. This is due to the nature of our original data source and the lack of other available datasets on 3-class face mask detection. Fortunately, this dataset has provided us with a feasible starting point for our subsequent application.

3. Explanation of the Methods Used

We come up with two methods to realize face mask detection. In the first method, we treat face localization and classification as two separate tasks. In the second method, we apply the YOLO V4 model to perform localization and classification simultaneously.

3.1. Face Localization + Image Classification

Figure 2. Localizer + Classifier



As shown in Figure 2, the input image will sequentially go through a face localization model and a classification model. The face localizer will crop out faces from the input image and pass them to a classifier. The classifier will then make predictions and classify these face images into three categories. In this project, we directly uses OpenCV Cascade Classifier [5] for face localization. We put more emphasis on the classification part.

3.1.1 Models

(a) Simple CNN

We started by constructing a simple convolutinoal neural network which consisted of two convolution layers, two max-pooling layers and two linear layers. This simple model helped us get familiar with the whole structure of our application and provided the steppingstone for us to implement more advanced neural networks.

(b) Transfer Learning

Since our dataset is relatively small, we then applied transfer learning for better performance. Two pre-trained models were chosen for comparison: the VGG16 and the ResNet50[1].

3.1.2 Loss Function

For the classification problem, we used the weighted cross entropy loss, considering the imbalance of our dataset. The ordinary loss function can

[5] https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

be described as[6] :

$$loss(x, class) = -log(\frac{exp(x[class])}{\sum_j exp(x[j])}) = -x[class] + log(\sum_j exp(x[j]))$$

Adding the weight to the loss function,

$$loss(x, class) = weight[class](-x[class] + log(\sum_j exp(x[j])))$$

where the weight is calculated as:

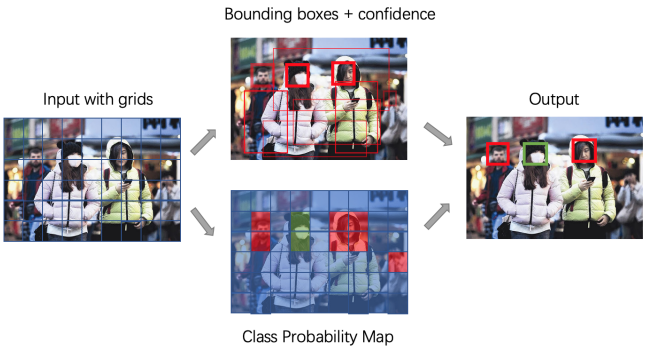
$$weight_i = 1 - \frac{\text{Size of Class}_i}{\text{Total Size}}$$

As the size of incorrectly masked class and unmasked class are significantly smaller, we put large weight to these two categories so that the model would put more emphasis on them.

3.2. YOLO V4

3.2.1 You Only Look Once

Figure 3. YOLO



You only look once (YOLO) is a state-of-the-art algorithm for real-time object detection known for its fast and accurate detection performance. Unlike prior detection methods that apply localizers or classifiers to the input images at multiple locations and scales, the YOLO approach merely applies a single neural network to the whole image. By dividing the image into grid regions, the model predicts bounding boxes and probabilities within each region. This enables the algorithm to fulfill the task of localization and classification altogether, which is one of the main reasons for us to choose this method for our project. Besides, we notice that there have not been any YOLO-based submissions in the Kaggle contest of face mask detection so far, and this gives us another reason to explore the YOLO method's performance regarding the task of face mask detection.

3.2.2 YOLO Loss

The loss function of the YOLO V4 model follows the 3-term loss function design of YOLO, which is defined as the sum of classification loss, localization loss, and confidence loss. [3]

(a) Classification Loss

When an object is detected, the classification loss of each grid cell is the squared error of the class conditional probabilities for each class:

$$\mathcal{L}_{classification} = \sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \quad (1)$$

where $\mathbb{1}_i^{obj} = 1$ if an object appears in cell i, otherwise 0, $\hat{p}_i(c)$ denotes the conditional class probability for class c in cell i [3].

(b) Localization Loss

The localization loss measures the errors in the predicted bounding box locations and sizes, and YOLO V4 replaces the MSE loss with the Complete IoU (CIOU) loss to speed up the bounding box regression.

$$\mathcal{L}_{CIOU} = 1 - IOU(A, B) + \frac{\rho^2(\hat{X}_{ctr}, X_{ctr})}{c^2} + \alpha \cdot v \quad (2)$$

where IOU the ratio of intersection over union, $\rho(.)$ denotes the Euclidean distance, $\hat{X}_{ctr}, X_{ctr}$ be the center of predicted/ground-truth centers of bounding boxes, c is the diagonal length of the minimum bounding box for predicted/ground-truth boxes, $\alpha = \frac{v}{(1-IoU)+v'}$, $v = \frac{4}{\pi^2} \cdot (\arctan \frac{w}{h} - \arctan \frac{\hat{w}}{\hat{h}})^2$.[7]

[6] https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

[7] https://zhuanlan.zhihu.com/p/159209199

(c) Confidence Loss

The confident loss measures the objectness of the box, and it's assigned computed with different formula when there an object is / is not detected in the box.

$$\mathcal{L}_{confidence} = \begin{cases} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_i^{obj} (C_i = \hat{C}_i)^2 \\ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_i^{noobj} (C_i = \hat{C}_i)^2 \end{cases}$$

where $\mathbb{1}_i^{noobj}$ is the complement of $\mathbb{1}_i^{obj}$, $\hat{C}_i$ is the box confidence score of the box j in cell i, λ_{noobj} weights down the loss while detecting background (default: 0.5) [3].

3.2.3 Using YOLO for face mask detection

Github project *yolov4-pytorch*[8] provides an implementation of YOLO V4, including the tricks like mosaic data augment, label smoothing, and learning rate annealing. We apply it to our 3-class face mask detection task based on the given framework for object detection.

To ensure the quality of face data for training, we filter the dataset and abandon the images with more than 12 faces. After splitting all the 785 images, we finally get 471 images for training, 157 images for validating, and 157 images for testing. On top of that, we adjust the given object detection framework of the YOLO V4 model that completes the localization and classification tasks simultaneously. Furthermore, we also use the pre-trained weights on COCO dataset[9] to speed up the model training process.

In general, we train our YOLO V4 model for 40 epochs, freezing the backbone for the first half and unfreeze those weights during the second half. While training our YOLO V4 models, we apply the recommended hyperparameter settings provided by the YOLO V4 object detection framework we found, and it performs quite fairly for the loss descending speed. Here, the first half of training sets the learning rate to $1e-3$ with a batch size of 4, and the second half sets the learning rate to $1e-4$ with a batch size of 2. After applying the cosine annealing learning rate[2] (weight_decay = $5e-4$, T_max=5, eta_min=$1e-5$), the training process of the entire model converges much faster and gives a satisfying performance.

Due to the complexity of YOLO models, it still takes around 40 minutes to train for 40 epochs on Google Colaboratory[10], even if both the YOLO V4 model and us applies various tricks to speed up the training process.

4. Results

With three classes in total, we set our accuracy baseline to be 79.5%, namely predicting all the inputs as "with_mask". In the following part, we will go through all the models we trained and compare them with each other in terms of accuracy.

4.1. Image Classification

Figure 4 and 5 shows the results of the simple CNN model for the best settings. As epochs increases, the loss decrease and the accuracy growth for the validation set slows down. Finally, we obtain the result with around 92% accuracy for the test set within 15 epochs.

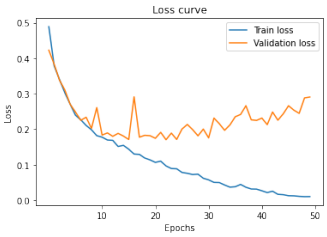


Figure 4. Simple CNN Loss

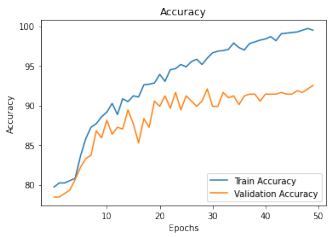


Figure 5. Simple CNN Accuracy

Figure 6 and 7 shows the results for the VGG16 model. We trained the model for 20 epochs. From the learning curve, we observe that after 5 epochs of training, the validation loss starts to go up while the training loss keeps decreasing, which is a sign of overfitting. Taking the number of epoch as a hyper-parameter, we limit the epoch number to 5 and re-train the model. The model ends up with an accuracy around 96% for the test set.

[8] https://github.com/bubbliiiing/yolov4-pytorch

[9] https://pjreddie.com/darknet/yolo/

[10] Google Colaboratory provides free usage of GPU, so we mainly train our YOLO V4 models on this platform. https://colab.research.google.com/

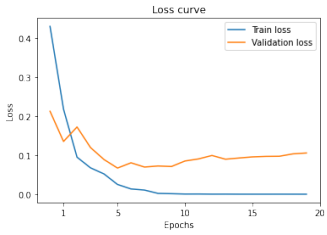


Figure 6. VGG16 Loss

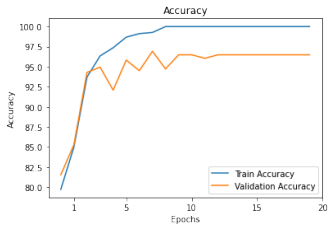


Figure 7. VGG16 Accuracy

The learning curve and the accuracy curve of the ResNet50 is similar to that of the VGG16 model. With a few epochs of training, the model shows signs of overfitting as we see a clear elbow point. After 5 epochs, the accuracy growth almost stops. We re-train the model and it ends up with an accuracy around 95% for the test set within 5 epochs .

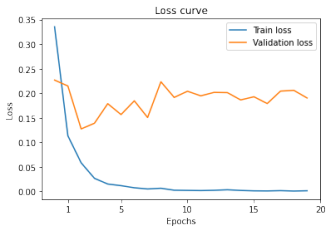


Figure 8. ResNet50 Loss

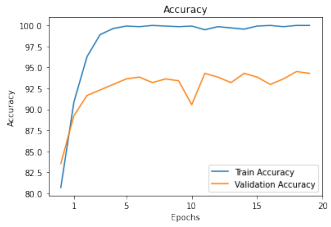


Figure 9. ResNet50 Accuracy

Figure 10, 11, and 12 show the normalized confusion matrices of the above models. Overall, all of the three models have similar performance and beat the baseline. The VGG16 model has lower average loss and performs slightly better than the two other models, but the overall improvement is insignificant. All of the models perform relatively poor on classifying 'incorrectly_masked'.

4.2. YOLO V4

4.2.1 Losses Curve

Figure 13 shows the results for training loss while using Adam or SGD as the optimizer. After training for 40 epochs, the both train and validation losses converges to around 10, which seems to be a nice fit.

4.2.2 Mean Average Precision

The mean Average Precision is a commonly used measure for object detection model accuracy, and we

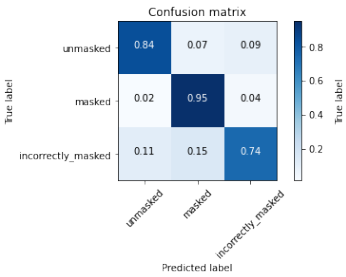


Figure 10. Simple CNN

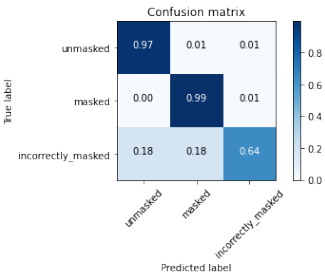


Figure 11. VGG16

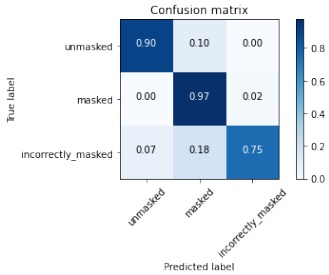


Figure 12. ResNet50

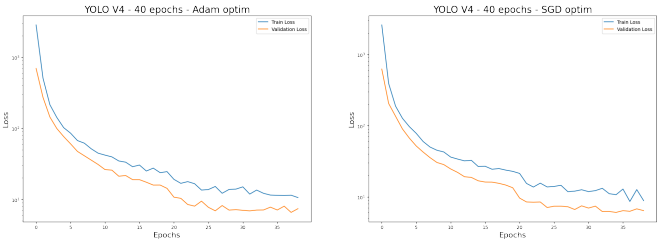


Figure 13. YOLO V4 Losses

use Cartucho's implementation[11] on Github to evaluate the performance of our own model.

First, the detection results are sorted by decreasing confidence and are assigned to the ground-truth objects. We define a detection result as true positive when it shares the same label with the ground-truth and the IoU (Intersection over Union) is greater than 0.5. In this process, all the ground-truth information only uses once to avoid the redundant counts of objects. We can then get the average precision values by plotting the precision/recall curve and computing the areas under the curve by numerical integration. Finally, we can get the mAP by taking an average. The results of our models are shown below. Figure 14 shows a comparison of the mAP measure, whereas Figure 15 shows our model's performance on the test set by class.

Unlike a pure classification that we can balance

[11] https://github.com/Cartucho/mAP

the class distribution by flipping, cropping, and rotating, it is almost impossible to deal with an imbalanced dataset when the task contains localizing the faces from a global image. The lack of mask_weared_incorrectly class data finally leads to poor performances in detecting this class. For our project, the model uses Adam optimization to make two correct predictions on the incorrectly masked case, whereas the model uses SGD optimization failed to detect any incorrectly masked faces. Apart from that, these two models share similar performances in detecting masked and unmasked faces. From an overall perspective, the model uses Adam optimization performs slightly better according to the mAP measure.

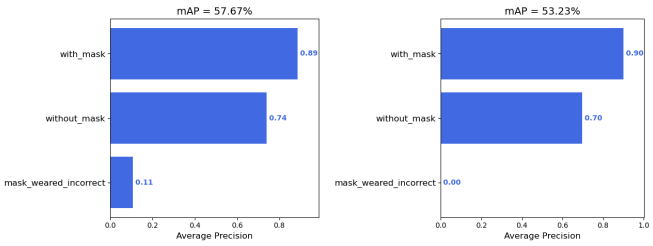


Figure 14. mAP Comparison (L:Adam; R:SGD)

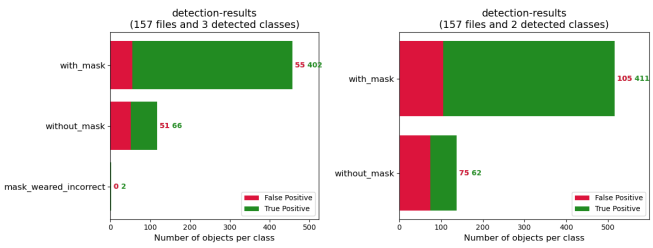


Figure 15. Detection Results (L:Adam; R:SGD)

We finally use the weights from the YOLO V4 model with Adam optimization to build our demo for face mask detection, and the results are consistent with the performance we get on the test dataset.

5. Conclusion and Future Work

Overall, this project explores two distinctive ways of performing face mask detection, and they all have their own advantages and shortcomings. For the integrated approach, we have more freedom to preprocess and rebalance the data, which enables us to give more accurate predictions on the classes with fewer data. For the YOLO V4 method, the model's final performance heavily depends on the distribution of classes,

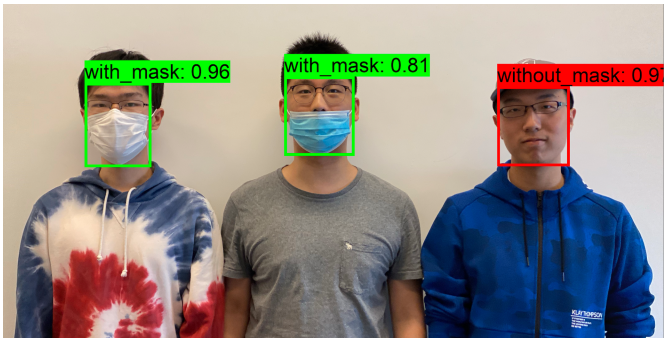


Figure 16. Face Mask Detection Demo

and the space of optimization is also limited due to the inconvenience of performing re-balancing operations. With such a severely unbalanced dataset, the YOLO model may hardly give accurate predictions. As for the aspect of face localization, the integrated approach is somehow restrained since the face detection model was trained on normal faces[4]. The face detection module of OpenCV does not have a consistent, satisfying performance on the masked data, which inevitably influences the effect of real-time detection. On this point. The YOLO method performs better and gives accurate localization results.

For the project, the imbalance of the dataset set some unsolvable constraints for the models' final performance. If we have enough time to dedicate to the face mask detection task, the most important task should be building a well-balanced dataset that is consistent with the intended application scenarios. All in all, the data is always the cornerstone of the models, at least for this field. Apart from that, we may also try some other state-of-the-art methods such as the faster R-CNN and make further comparisons regarding different aspects, such as efficiency, accuracy, and computational power.

6. Acknowledgements

This project is tutored by Prof. Enric Junque de Fortuny and Instructor Zijian Zhou in CSCI-SHU 360 Machine Learning (Fall 2020) at NYUSH.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[2] I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

[3] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[4] J. M. Viola, P. Robust real-time face detection. *International Journal of Computer Vision*, 57:137–154, 2004.