Facemask Detector

TRINH Van Hoan, NGUYEN Kha Nhat Long
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
{vhtrinh, knlnguyen}@connect.ust.hk

Abstract

Along with the growth of society, many new problems emerge and need our attention. Among them, there are problems that need urgent solutions due to their critical nature. Monitoring mask-wearing in public areas is such a problem, especially when mask-wearing is a key solution for the COVID pandemic in recent years.

Many solutions that focus on monitoring public areas, mainly about whether they wear masks or not, have been proposed. Unfortunately, they share the same drawbacks as heavily relying on human supervisors both for detecting and reporting the incidents.

In this report, we implement an approach for autonomous monitoring that deploys an AI system on images or live video retrieved from a camera in a public area. It consists of two components, a Multi-Face Detector, and a Multi-Class Classifier. Here we implement and train the State-of-the-art model for each subtask, MTCNN, and EfficientNet respectively. The performance of the pipeline on the test set will be reported for evaluation.

Overall, our AI model achieves lower but close to the reported performance of two components in their original papers. Moreover, when running live with live video, the model can have satisfying results while being able to follow the video stream continuously.

1. Introduction

1.1. The problem

In recent years, we have seen how large a respiratory virus (COVID) can disrupt human society as a whole, and how wearing masks, a simple solution, can effectively help us to contain the disease from spreading. Therefore, being able to regularize and monitor this mask mandate in a correct, fast, and large-scale manner is of paramount importance.

This is a new and challenging problem given its meanings and crucial applications, since solving this problem can help us to deal not just with COVID but with other similar problems as well. The two main key points to solve this problem are to produce predictions correctly and be able to run fast. These also are the two goals that our project aims to achieve.

1.2. Related works

When the pandemic first came out, the idea to develop an Artificial Intelligence tool to automatically detect people wearing masks was already proposed. However, it did not work out due to the lack of relevant data to train on. Therefore, in the early days, the most common method that was applied in many places is to have real people manually monitor the cameras and send signals to the system when there is someone not wearing a mask properly.

Another way that is being developed recently is to build an end-to-end Machine Learning model to take a live video as an input and output the detections. However, the drawback of this approach is that labeling the faces, and masks, and classifying them in every frame is very costly.

1.3. Our approach

With the same idea but attempting to overcome those drawbacks, this project will focus on building an AI monitor tool to detect faces and classify the correctness of wearing masks from frames retrieved from live videos, adopting State-of-the-art (SOTA) Object Detection and Image Classification models.

In this project, we have implemented the architecture for both the Detector and Classifier and trained both using public datasets. Along the process, we always aimed for the model to be lightweight in order to produce predictions in real-time, as well as produce satisfying results.

In the next sections, we will discuss the datasets we worked on, the architecture we chose for the two subtasks and report its performance on the test set in terms of different loss functions for the Detector, and in terms of the accuracy, precision, recall, and F1 score metrics for the Classifier. Finally, we will measure the model under a live video setting.

2. Datasets

2.1. To train the Detector

We searched for a public face dataset and chose the WiderFace dataset[3] to be suitable for our project in terms of both the variety and scale of the data.

This dataset contains 32203 images from 61 different classes of events. The annotations include the bounding boxes for 393703 faces in total with a high degree of variability in scale, pose, and occlusion. Some samples of this dataset can be found in Figure 1.

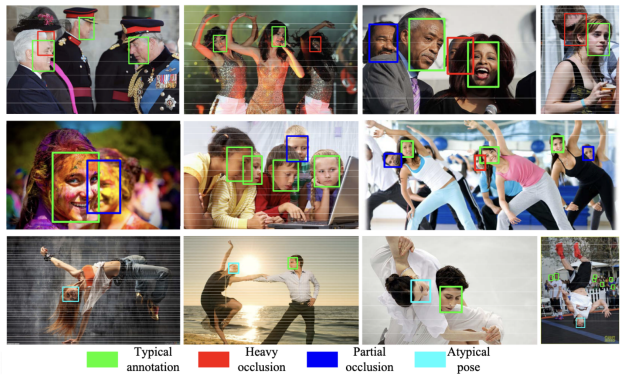

Figure 1: Sample images from the WiderFace dataset

2.2. To train the Classifier

We searched for a public dataset that is suitable for our classifier, and the one we chose is the **Face Mask Dataset** on Kaggle

This dataset has a total of 13000 images, which consists of 3 classes: correct mask (4000 images), incorrect mask (5000 images), without mask (4000 images). The images were taken and processed from multiple sources: standard datasets such as MAFA, MFDD, from the internet, and by simulations in order to be diverse and unbiased. Figure 2 shows some samples from this dataset. We then shuffled and splited it into training, validation, test set (11000-1000-1000) to train and evaluate the model. The performance on the test set will be reported in the final report.

Figure 2: Sample images from the Face Mask dataset

3. Methodology

As mentioned previously, the AI monitor consists of two major components:

1. A multi-face Detector to locate every faces inside the input frame.

2. A multi-class Image Classifier to determine whether the faces are wearing facemasks correctly, incorrectly, or not wearing at all.

3.1. Detector (MTCNN)

Face detection is challenging due to various poses, illuminations and occlusions. In this project, we implemented a deep-learning architecture to find all the positions of faces as bounding boxes in a frame. To achieve satisfying results as well as being fast, we chose to implement MTCNN[2] (Multi-Task Cascaded Convolutional Networks), a deep cascaded multi-task framework which adopts a cascaded structure with multiple stages of deep convolutional networks.

We use this GitHub repository as the base to implement the MTCNN architecture from scratch. In short, the MTCNN model utilizes different CNN-based architectures to perform three tasks: face classification, bounding box regression, and landmark regression.

To achieve the objective of our pipeline, we use the WiderFace dataset[3] to generate the data to train the model. We only focus on only the first 2 sub-tasks: face classification (predict if that sample is a face) and bounding box regression (predict the locations of the bounding boxes).

The idea of MTCNN is similar to R-CNN architecture[1] in that it uses CNN to generate Region Proposals for training. The MTCNN model contains three stages: PNet, RNet, and ONet that is described in Figure 3. Each stage's outputs are used to generate inputs for the next stage.

3.1.1 Preprocess Data

At first, each image in the dataset is duplicated to different sizes to build an Image Pyramid. In particular, we resized the image into three different sizes: 12x12, 24x24, and 48x48, and produced 96609 images from 32203 original images.

Before training each network (except the first one), we used the network from the previous stage to generate Regional Proposals for the next stage. This process is time costly and costs a lot of storage that requires carefully managing all the intermediate results in the local machine.

3.1.2 PNet

At the beginning of training PNet, a randomized algorithm was used to create candidate bounding boxes. The

generated box is of size 12x12. We generated over 1000000 samples to train the PNet from the original WiderFace dataset.

The PNet architecture is shown in Figure 4. It is a CNN architecture with three layers with filter size 3x3.

3.1.3 RNet

We used the previous model PNet to generate candidate bounding boxes for the RNet. Over 700000 samples were created using the WiderFace dataset and the PNet model.

The RNet architechture is shown in 5. It also uses CNN architechture. There are four layers in RNet with two CNN layers using filter size 3x3, and one layer using filter size 2x2. The last layer is a fully connected layer.

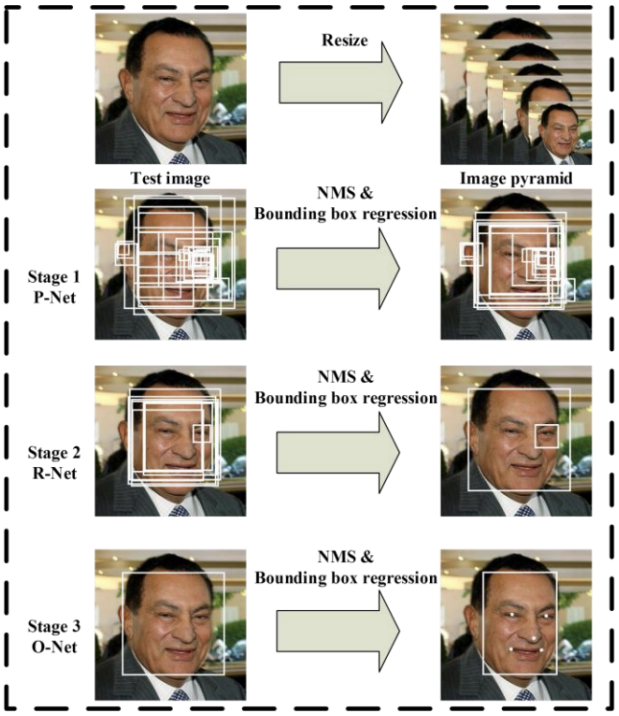

Figure 3: MTCNN Architecture

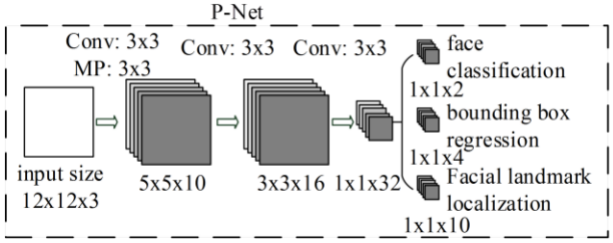

Figure 4: PNet Architecture

3.1.4 ONet

The ONet architecture is shown in 6. Similarly to the RNet, before training the ONet, we also used the WiderFace dataset and the intermediate results from both the PNet and the RNet to generate training data. We generated over 300000 images of size 48x48 as inputs for the ONet.

The ONet contains five layers with three CNN layers using filter size 3x3, and one CNN layer using filter size 2x2. Lastly, it uses fully connected layers to produce the final predictions.

3.1.5 Training Process

Loss Function:

1) Face Classification

We use the cross-entropy loss for the Face Classification task:

$$L_i = -(y_i log(p_i) + (1 - y_i)(1 - log(p_i))) \quad (1)$$

where p_i is the probability generated by the network that indicates a sample being a face. The notation $y_i \in \{0, 1\}$ denotes the ground-truth label.

2) Bounding Box Regression

For each candidate window, we predict the offset between it and the nearest ground truth (i.e., the bounding boxes' left, right, up, and down). The learning objective is formulated as a regression problem, and we employ the Euclidean loss for each sample:

$$L_i^{box} = ||y_i^* - y_i||^2 \quad (2)$$

where y_i^* is the target generated by the network and y_i is the ground-truth coordinates. There are four coordinates, including left, right, up, and down, thus $y_i \in R^4$.

Accuracy Function:

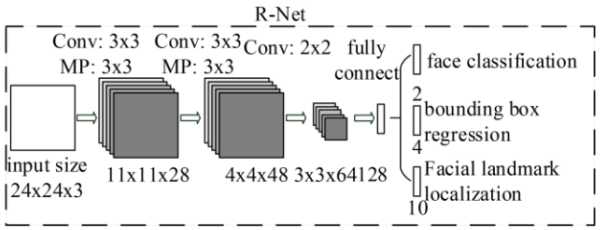

Figure 5: RNet Architecture

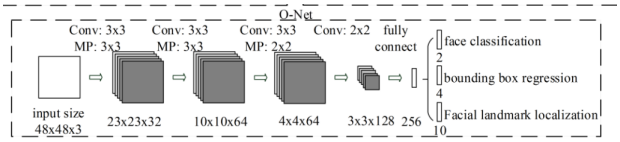

Figure 6: ONet Architecture

1) Training accuracy equation:

$$acc_i = \frac{c^i_{area}}{g^i_{area}} \tag{3}$$

where c^i_{area} is the common area between generate bounding box and the original bounding box and g^i_{area} is the area of generating a bounding box

2) Testing accuracy function:

We use the below function to evaluate the model.

$$acc_i = 1 - \sqrt{\frac{((|l^*_i - l_i| + |r^*_i - r_i|)(|u^*_i - u_i| + |d^*_i - d|)}{(|l_i + r_i|)(|d_i + u_i|)}} \tag{4}$$

where l_i, r_i, u_i, d_i are the left, right, up, and down coordinates of the ground-truth bounding box, and $l^*_i, r^*_i, u^*_i, d^*_i$ are the left, right, up, and down coordinates of the bounding box generated by the network.

3.2. Classifier (EfficientNet)

After the faces' positions are predicted, the bounding boxes will subsequently be cropped and fed into the image classifier to make inferences. Each cropped box will be classified into three classes: wear masks correctly, incorrectly, or not wearing any at all.

The model we chose for classification is EfficientNet[4]. EfficientNet is a family of convolutional neural network mainly for classification. Its primary aim is to have better paramater efficiency and faster training speed, i.e. to scale up the model efficiently while maintaining the performance. Using Neural Architecture Search (NAS) method, this family of models is designed and scaled up to achieve much better accuracy and efficiency than previous ConvNets.

Although the author has released the 2nd version EfficientNetV2[5], after trying to build both of them, we selected the base model of EfficientNet, EfficientNet-B0 because it is lightweight (has only 4M parameters in our implementation while EfficientNetV2-S has 22M parameters). It is fast to train and able to run in real-time while having a high performance. Figure 7 shows the architecture of EfficientNet-B0.

As can be seen in Figure 8, at its release, in comparison to other SOTA models, EfficientNet-B7 achieves state-of-the-art 84.3% top-1 accuracy on ImageNet, while being 8.4x smaller and 6.1x faster on inference than the best existing ConvNet. EfficientNets also transfer well and achieve state-of-the-art accuracy on CIFAR-100 (91.7%), Flowers (98.8%), and 3 other transfer learning datasets, with an order of magnitude fewer parameters.

4. Experimental Results

4.1. Detector

Implementing and training

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	40	2
5	MBConv6, k3x3	28 × 28	80	3
6	MBConv6, k5x5	14 × 14	112	3
7	MBConv6, k5x5	14 × 14	192	4
8	MBConv6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

Figure 7: EfficientNet-B0 architecture: each row describes stage i with $\hat{L}i$ layers, input resolution $<\hat{H}i, \hat{W}i>$, output channels $\hat{C}i$

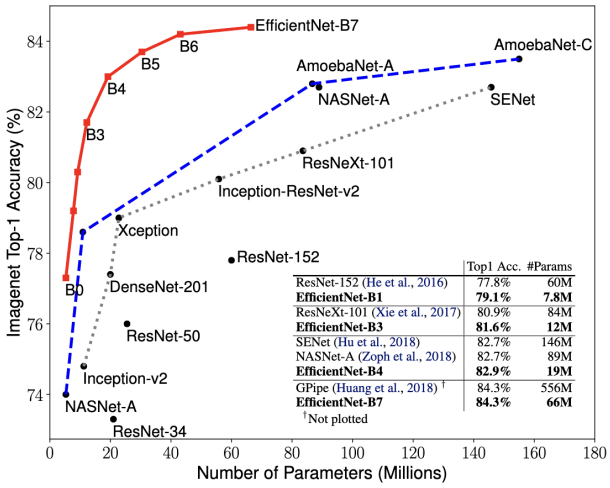

Figure 8: Model Size vs. ImageNet Accuracy

We followed the network architecture of MTCNN and tune the parameter for each sub-network and the generated data for each network. We tried different batch sizes (16,32,128) and grid search the learning rate from (0.05 to 0.005).

The optimal configuration is to have the detector weights updated by the Stochastic Gradient Descent (SGD) optimizer with the following parameters: learning rate 0.01, frequent 200, and batch size 128 for each sub-network.

Result

To ensure that the training process progresses in the right direction, we kept track of the loss after every epoch throughout the whole process. Other than the loss, we also saved checkpoints for the weights of the detector and selected the best checkpoint corresponding to the epoch that has the smallest loss for making inferences and evaluation. With the weights loaded from that checkpoint, the detector performs reasonably well. The quantitative results on the test set is shown in Table1.

Model	Accuracy	Bounding box	Classification
PNet	91%	0.03	0.15
RNet	92%	0.02	0.1
ONet	**94%**	**0.002**	**0.0001**

Table 1: Performance of MTCNN using the above-mentioned loss functions

4.2. Classifier

Implementing and training

Based on its original paper, we implemented the EfficientNet-B0 architecture as the classifier and then experimented with different settings to train the classifier. We tried different batch size values (16, 32, 64), and grid-searched based on the log space for the learning rates (0.1 to 0.0001), momentum (0 to 0.9), and weight decay (0.01 to 0.0001).

The optimal configuration for classifier training is updating the weights by the Cross-Entropy Loss function and Stochastic Gradient Descent (SGD) optimizer with the following parameters: learning rate 0.001, momentum 0.9, and weight decay 0.0005. For each training session, we loaded the previously saved weights and trained for a few epochs, observed its performance (on the validation set), and saved the best checkpoint for the later sessions. An example of a training session can be found in Figure 9.

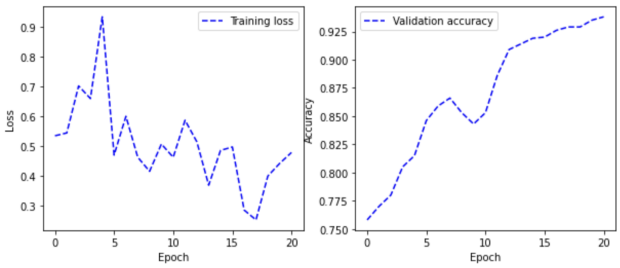

Figure 9: An example of a training session (start at epoch 30)

Results

We kept track and selected the model checkpoint corresponding to the epoch that has the highest validation accuracy for making inferences and evaluation. With the weights loaded from those checkpoints, the classifier instance performs well and its statistics on the test set can be seen in Table 2.

4.3. The whole pipeline

Overall, both detection and classification models yield sufficiently high performance and meet our original goal.

	Precision	Recall	F1	Support
correct	0.847	0.996	0.915	300
incorrect	0.997	0.867	0.927	400
not wearing	0.996	0.993	0.995	300
accuracy			0.944	1000
macro avg	0.946	0.952	0.946	1000
weighted avg	0.952	0.944	0.944	1000

Table 2: Performance of the classifier on the test set

Figure 10 shows the sample predictions from the project demo[1].

Figure 10: Left: sample output of the detector. Right: sample output of the whole pipeline

Live video setting

To measure its usefulness in practice, we measure the model under a live video setting. We use a camera to take a video stream as input. Then at each loop, we constantly run each input frame into our pipeline and create an overlay image that contains the bounding boxes and the labels, and then overlay the bounding box image back onto the next frame of the video stream to display and continue this process.

Testing the model under this setting, our model can produce a satisfying performance in multi-face detection and image classification. Each loop runs in approximately 1.5 seconds on CPU, and approximately 0.5 seconds on GPU, which means it can be used for a live camera in practice, achieving our goal for the project.

Code and demo of the project can be found at this[1] Github repository.

5. Conclusion

In this project, we have implemented and trained an autonomous AI supervisor to help monitor mask-wearing successfully. The system consists of two main components,

[1] https://github.com/VanHoann/Facemask_Detector

a multi-face detector, and a multi-class image classifier instance. The detector, which is MTCNNN, is responsible for locating and correspondingly extracting spatial information of every faces inside input frames. Subsequently, single-face images will be cropped out based on that information and be classified by the classifier instance, which is EfficientNet, if their according face is wearing masks correctly, incorrectly, or not wearing any. In the live video setting, taking the video stream as input, the model outputs can be displayed respectively by the predefined color of the bounding boxes and labels in the output frames in a fast manner. We believe that our AI monitor can be extended and applied to many more use cases not limited to monitoring mask-wearing.

Further improvement

During manual testing, we observed that some faces are not detected or incorrectly classified by the system when they are too small within the frame, under relatively low-light conditions, or surrounded and heavily blocked by some foreground objects. To solve this case, we will need to train with a larger and more challenging dataset. In the future, we want to focus on this aspect to further improve the pipeline.

References

[1] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[2] Z. L. Kaipeng Zhang, Zhanpeng Zhang and Y. Qiao. Joint face detection and alignment using multi-task cascaded convolutional networks, 2016.

[3] C. C. L. Shuo Yang, Ping Luo and X. Tang. Wider face: A face detection benchmark, 2015.

[4] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[5] M. Tan and Q. V. Le. Efficientnetv2: Smaller models and faster training, 2021.