Facemask detection model using MMdetection

Mukul Kumar Vishwas
Department of Mathematics and Mechanics
Novosibirsk State University
Novosibirsk, Russia
m.vishvas@g.nsu.ru

Petr Menshanov
Novosibirsk State University
Novosibirsk, Russia

Aleksey Okunev
Novosibirsk State University
Novosibirsk, Russia

***Abstract*— this paper described a face mask detection model using MMdetection toolbox which predicts if the person on the picture wore mask correctly or incorrectly or not, in current situation this model is very useful as this simple precaution will help to stop spreading of deadly Coronavirus.**

Keywords— MMdetection, predicts, coronavirus

I. INTRODUCTION

The COVID-19 pandemic also widely known as coronavirus pandemic caused by Severe Acute Respiratory Syndrome Coronavirus 2 (SARS_Co_2). The World Health Organization declared the outbreak a Public Health Emergency of International Concern on 30 January, and a pandemic on 11 March[1], It completely changed our life and cost more than 393 thousand of lives.

On the other hand by following some simple rules (mentioned below) we can stop spreading of this disease:

A. *Face masks and respiratory hygiene.*

B. *Social distancing.*

C. *Self-isolation.*

The goal of this work is to create and train a neural network to discriminate peoples who follow the sanitary rules like wearing the face mask properly from those people who are violating them.

It should able to return a boundary box around the face describing three things:

A. *Tag on the face which describe if the person wore the mask properly or not.*

B. *A boundary region with the probability/confidence of the model's prediction.*

II. MMDETECTION

A. What is MMdetection

MMdetection is an object detection toolbox that contains a rich set of object detection and instance segmentation methods as well as related components and modules[2].

Major features of MMdetection are:

1) ***Modular design:*** *The detection framework is decomposed into different components and one can easily construct a customized object detection framework by combining different modules.*

2) ***Support of multiple frameworks:*** *The toolbox supports popular and contemporary detection frameworks like Fast R-CNN, Faster R-CNN, Mask R-CNN, RetinaNet, DCN etc.*

3) ***High efficiency:*** *All basic bbox and mask operations run on GPUs. The training speed is faster than or comparable to other code–bases including Detectron, mask rcnn-benchmark and SimpleDet.*

It also provides vides weight of more than 200 network model.

B. Archetecture

Although the model architectures of different detectors are different, they have common components, which can be roughly summarized into the following classes:

1) ***Backbone:*** *Backbone is the part that transforms an image to feature maps, such as a ResNet-50 without the last fully connected layer.*

2) ***Neck:*** *Neck is the part that connects the backbone and heads. It performs some refinements or reconfigurations on the raw feature maps produced by the backbone. An example is Feature Pyramid Network (FPN).*

3) ***Dense Head (Anchor Head/ Anchor Free Head):*** *Dense Head is the part that operates on dense locations of feature maps, including Anchor Head and Anchor Free Head, e.g., RPNHead, RetinaHead, FCOSHead.*

4) ***RoIExtractor:*** *RoIExtractor is the part that extracts RoI-wise features from a single or multiple feature maps with RoIPooling-like operators. An example that extracts RoI features from the corresponding level of feature pyramids is single RoIExtractor.*

5) ***RoIHead (BBoxHead/ MaskHead):*** *RoIHead is the part that takes RoI features as input and makes RoI-wise task specific predictions, such as bounding box classification/ regression, mask prediction.*

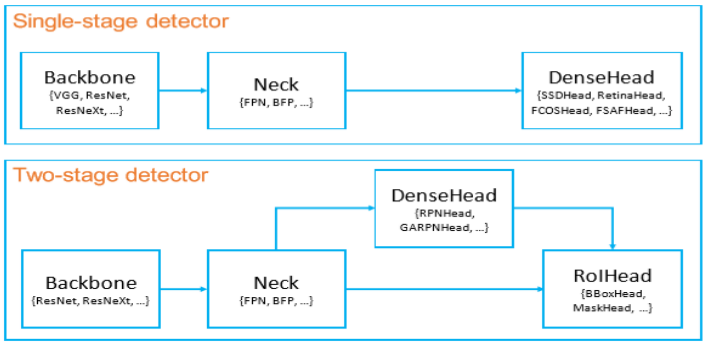


Fig 1: Framework of single-stage and two stage detector

C) Resnet50 with and without FPN

In this section we will look into the architecture of the object detection with and without feature pyramid network (FPN), Figure 2 describing the architecture of object detection without FPN.

Detecting object in different scale and size is a challenging task to overcome this issue we can use same image with different size/scale but this approach has some disadvantages like high memory demand and high time consumption [3].

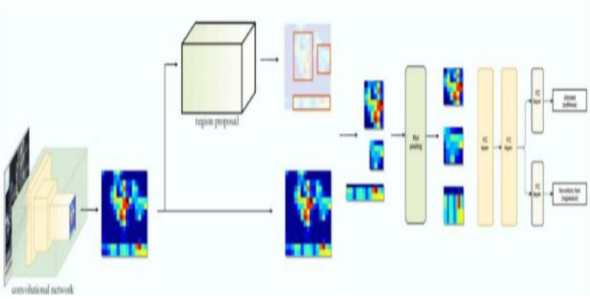


Fig 2: Object detection without FPN.

FPN has a beautiful solution for the above issue, it create a pyramid of feature and use them for object detection as showed in figure 3. Feature Pyramid Network (FPN) is a feature extractor designed of pyramid concept with accuracy and speed in mind.

It replaces the feature extractor of detectors like Faster R-CNN and generates multiple feature map layers (multi-scale feature maps) with better quality information than the regular feature pyramid for object detection. FPN composes of a bottom-up and a top-down pathway. As we go up, the spatial resolution decreases. With more high-level structures detected, the semantic value for each layer increases.

FPN extracts feature maps and later feeds into a detector, says RPN, for object detection. RPN applies a sliding window over the feature maps to make predictions on the object (has an object or not) and the object boundary box at each location[3].

In the FPN framework, for each scale level a 3 × 3 convolution filter is applied over the feature maps followed by separate 1 × 1 convolution for object predictions and boundary box regression. These 3 × 3 and 1 × 1 convolutional layers are called the RPN head. The same head is applied to all different scale levels of feature maps.

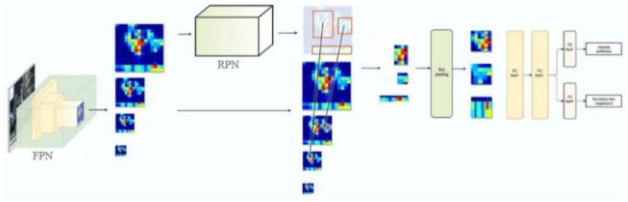


Fig 3: Object detection with FPN.

D) Formula to pick feature map

The formula to pick the feature maps is based on the width w and height h of the ROI.

$$k = \lfloor k_0 + \log_2(\sqrt{wh}/224) \rfloor.$$

Where,

ko = 4

k is the Pk layer in the FPN used to generate the feature patch.

So if k = 3, we select P3 as our feature maps. We apply the ROI pooling and feed the result to the Fast R-CNN head (Fast R-CNN and Faster R-CNN have the same head) to finish the prediction.

E) Comparision

The comparison of different feature is mentioned in the table 1.

Feature	**Without FPN**	**With FPN**
Training Time	Normal	Increased
Dataset requirement	Big	Small
Test/validation time	High	Low
Accuracy	Good	Increased
AR	44.9	56.3
Inference time	0.32 sec.	0.148 sec.

Where:

AR (Average recall): The ability to capture Object.

Inference time: Time taken for prediction.

III. MODEL

A) Model composition

In this section, we will see the model used for mask detection. In every section we can see the content from which that part was made of for example backbone was made of Resnet50 and followed by the neck which is a feature pyramid network (FPN).

*1) **Backbone:** The discription of the Resnet50 was described below figure:*

```
backbone=dict(
type='ResNet',
depth=50,
num_stages=4,
out_indices=(0, 1, 2, 3),
frozen_stages=1,
norm_cfg=dict(type='BN',    requires_grad=True),
norm_eval=True,
style='pytorch'),
```

*2) **Neck:** The neck used in this model is FPN and the full discription is below:*

```
Neck =dict(
type='FPN',
in_channels=[256, 512, 1024, 2048],
out_channels=256,
num_outs=5),
```

3) ***Dense head:*** *For the dense head we used RPNHead it extract feature from the dense part of the image:*

```
rpn_head=dict(
type='RPNHead',
in_channels=256,
feat_channels=256,
```

4) ***ROI head:*** *ROI head is a very interesting part of the model we used CascadeROIHead in this model and it prapose the region of interest from where the model detect the object and the discription is below:*

```
roi_head=dict(
type='CascadeRoIHead',
num_stages=3,
```

5) ***Optimizer:*** *We used Stochastic gradient descent (SGD) optimizer for our model. SGD is an iterative method for optimizing an objective function with suitable smoothness properties (e.g. differentiable or subdifferentiable). It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data). Especially in high-dimensional optimization problems this reduces the computational burden, achieving faster iterations in trade for a lower convergence rate[4].*

6) ***Learning rate****: Learning rate for our model was set to 0.02 for the training.*

7) ***Process of Training:*** *In this section we will see simple flow of the data through training. We can see all the main operation performed on data before feeding to the model.*

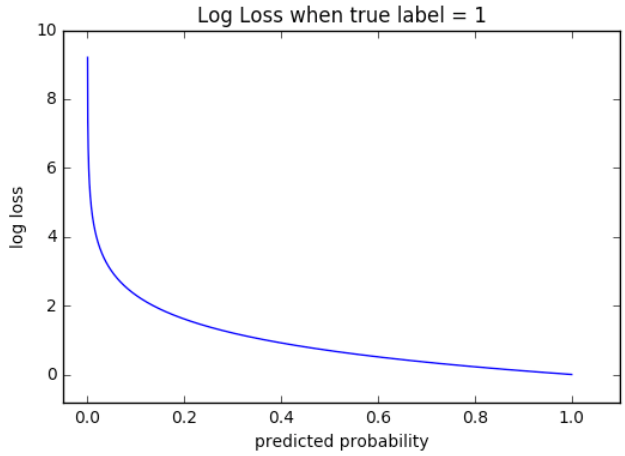
```
train_pipeline = [
  dict(type='LoadImageFromFile'),
  dict(type='LoadAnnotations', with_bbox=True),
  dict(type='Resize',img_scale=(1333,800),
keep_ratio=True),
  dict(type='RandomFlip', flip_ratio=0.5),
  dict(type='Normalize', **img_norm_cfg),
  dict(type='Pad', size_divisor=32),
  dict(type='DefaultFormatBundle'),
  dict(type='Collect',keys=['img','gt_bboxes','gt_labels'],
```

8) ***Process of Testing:*** *Here we can see all the operation performed on the testing data.*

```
train_pipeline = [
  dict(type='LoadImageFromFile'),
  dict(type='MultiScaleFlipAug',img_scale=(1333, 800),
      flip=False,
   transforms=[
     dict(type='Resize', keep_ratio=True),
     dict(type='RandomFlip'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
          dict(type='ImageToTensor', keys=['img']),
         dict(type='Collect', keys=['img']),
               ])]
```

B) Model loss during training

We used Cross Entropy Loss for our model. Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. So predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0[5].



The graph above shows the range of possible loss values given a true observation (Masked = 1). As the predicted probability approaches 1, log loss slowly decreases. As the predicted probability decreases, however, the log loss increases rapidly. Log loss penalizes both types of errors, but especially those predictions that are confident and wrong!

Cross-entropy and log loss are slightly different depending on context, but in prediction when calculating error rates/probability between 0 and 1 they resolve to the same thing.

Figure 3 showing epochs versus loss value graph for our model, we trained our model for 200 epochs. The loss value was taken on every 50 epochs to draw.

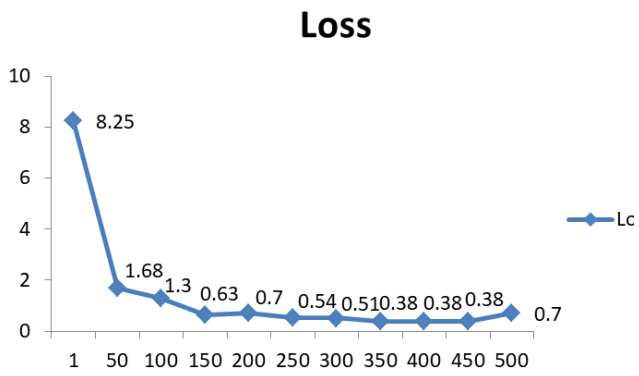


Fig 3: Loss during training

IV. Image Annotations

To train the model we need annotated images so our model can extract feature from the images and distinguish between masked and unmasked face. To train and validate the model we used VGG annotation, it is a very simple annotation tool which runs on your web browser.

We need the images to be in Common Objects in Context (COCO) formats; it stores the annotation details for the bounding box in JSON format. Main component of coco file are:

A) ***Info:*** *contains high-level information about the dataset.*

B) ***Licenses****: contains a list of image licenses that apply to images in the dataset.*

C) ***Categories****: contains a list of categories. Categories can belong to a super category.*

D) ***Images:*** *contains all the image information in the dataset without bounding box or segmentation information. image id's need to be unique.*

E) ***Annotations:*** *list of every individual object annotation from every image in the dataset.*

```
{
"info": info,
"licenses": [licenses],
"categories": [categories],
"images" : [images],
"annotations":[ annotations]
}
```

Above is an example of coco format.

Fig 4: Annotated face from left to right No mask, No mask, Masked, Masked, No mask, No mask

V. Result

In the initial phase this model was implemented using "cascade_mask_rcnn_x101_64x4d_fpn_20e_coco.py". In the figure 5 we can see the final output, the model is successfully able to predict and discriminate between Masked and unmasked faces Mask face is further classified to Mask and Incorrect.

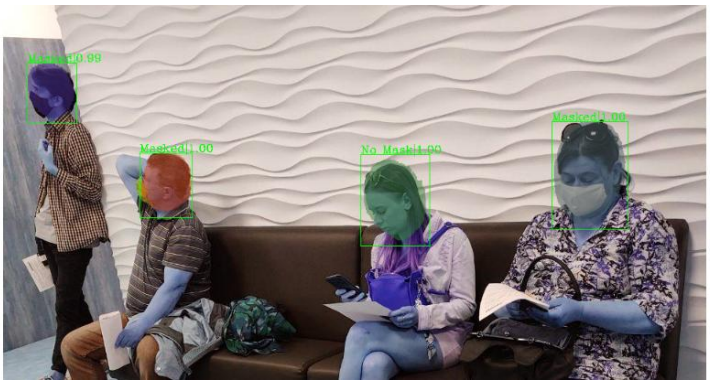


Fig: 5.1 final output of the model

Fig 5.2: Final output of the model

To measure the accuracy of our model we used mAP (mean Average Precision), it is a popular metric to measure object detector like faster R-CNN, SSD etc. It gives a calculated value between 0 and 1.
Calculating accuracy for an object detector is little complicated as we have to detect the object or class and also the area where the object was detected.

Precision: It calculates the Accuracy of the prediction.

Recall: It measures how good model find all the positives.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

TP = True positive

TN = True negative

FP = False positive

FN = False negative

Fig 5.3: Formula for Precision and Recall

Another important term we have to understand is **IoU (**Intersection over Union)[6], To check the correctness of our model's we first have to *judge the correctness of each of these detections*. The metric that tells us the correctness of a given bounding box is the IoU.

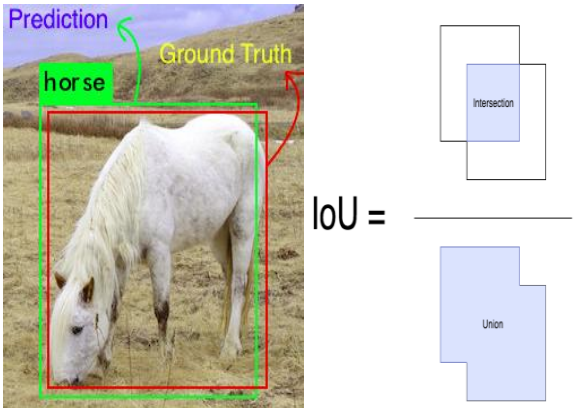


Fig 5.4: Visual representation of IoU.

To get TP and FP we use IoU, we now have to identify if the detection (a Positive) is correct (True) or not (False). The most commonly used threshold is 0.5 - i.e. If the IoU is > 0.5, it is considered a **True Positive**, else it is considered a **false positive**.

Matric	IoU	Old	New
mAP	@[IoU=0.50:0.95]	0.215	0.436
mAR	@[IoU=0.50:0.95]	0.228	0.531

Fig 5.s: mAP and mAR of the detection model

VI. FUTURE TASK

We are working to add below feature in our model:

A) *Train the model with more data to increase the accuracy.*
B) *Able to identify second point on live video streaming.*
C) *Feature space representation of each person so we can identify them and track on live feeding.*
D) *Quantization of the network so it will decrease power consumption and can run of small devices like Jetsn.*

REFERENCES

[1] Naming the coronavirus disease (COVID-19) and the virus that causes it- url: https://www.who.int/emergencies/diseases/novel-coronavirus-2019/technical-guidance/naming-the-coronavirus-disease-(covid-2019)-and-the-virus-that-causes-it

[2] MMdetection : Open MMlab detection toolbox and benchmark- url: https://arxiv.org/abs/1906.07155.

[3] Understanding Feature Pyramid Networks for object detection (FPN)- url: https://medium.com/@jonathan_hui/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c.

[4] Stochastic gradient descent –url: https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

[5] Cross Entropy Loss-url: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html.

[6] Measuring Object Detection models-mAP-What is Mean Average Precision? url: https://tarangshah.com/blog/2018-01-27/what-is-map-understanding-the-statistic-of-choice-for-comparing-object-detection-models

[7] mAP (mean Average Precision) for Object Detection url: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173