Transactions of the Indian National Academy of Engineering (2020) 5:509–518
https://doi.org/10.1007/s41403-020-00157-z

ORIGINAL ARTICLE

MOXA: A Deep Learning Based Unmanned Approach For Real-Time Monitoring of People Wearing Medical Masks

Biparnak Roy[1] · Subhadip Nandy[1] · Debojit Ghosh[1] · Debarghya Dutta[2] · Pritam Biswas[1] · Tamodip Das[2]

Received: 5 May 2020 / Revised: 1 July 2020 / Accepted: 11 July 2020 / Published online: 25 July 2020

Abstract
With 6.93M confirmed cases of COVID-19 worldwide, making individuals aware of their sanitary health and ongoing pandemic remains the only way to prevent the spread of this virus. Wearing masks is an important step in this prevention. Hence, there is a need for monitoring if people are wearing masks or not. Closed circuit television (CCTV) cameras endowed with computer vision function by embedded systems, have become popular in a wide range of applications, and can be used in this case for real time monitoring of people wearing masks or not. In this paper, we propose to model this task of monitoring as a special case of object detection. However, real-time scene parsing through object detection running on edge devices is very challenging, due to limited memory and computing power of embedded devices. To deal with these challenges, we used a few popular object detection algorithms such as YOLOv3, YOLOv3Tiny, SSD and Faster R-CNN and evaluated them on Moxa3K benchmark dataset. The results obtained from these evaluations help us to determine methods that are more efficient, faster, and thus are more suitable for real-time object detection specialized for this task.

Keywords Object detection · Face mask detection · COVID-19 · Deep learning

Abbreviations

mAP@50	Mean average precision with 50% IoU
IoU	Intersection over union
FPS	Frames per second
API	Application program interface
SGD	Stochastic gradient descent
FLOPs	Floating point operations

✉ Biparnak Roy
roybiparnak@gmail.com

Subhadip Nandy
subhadipnandyofficial@gmail.com

Debojit Ghosh
ghoshdebo2000@gmail.com

Debarghya Dutta
debarghyadutta2332k@gmail.com

Pritam Biswas
emailtobiswas2000@gmail.com

Tamodip Das
iamtamodip@gmail.com

[1] Instrumentation and Electronics Engineering, Jadavpur University, Kolkata, India

[2] Electrical Engineering, Jadavpur University, Kolkata, India

Introduction

As suggested by medical practitioners, it is a necessity to wear a mask whenever you are at public places or places with high probability of people gathering, to prevent the spread of the virus. Because it is so important to wear masks it is essential to have a monitoring system. Till now the only way of monitoring the people if they are wearing the mask was manual. Hence to overcome this cumbersome process of manual monitoring, we propose a real-time monitoring of people wearing medical masks with the help of artificial intelligence equipped with deep learning. We have come up with an easily adaptable model that can detect the people wearing masks from the images or live feed fed to it. This can be used in places such as banks, hospitals, transportation hubs, and even some traffic monitoring systems come with closed circuit television (CCTV) cameras pre-installed. These are areas which receive a high density of footfall, and thus become active sources of transmission. It is absolutely essential that people use medical masks in these areas.

Now the task of monitoring people wearing masks or not is particularly challenging because of the following reasons: (1) Detection of faces from a CCTV camera feed is particularly difficulty as their size is very small as compared to the rest of the environment (2) While wearing masks much of the facial features gets covered, therefore facial detection becomes very difficult (3) There is no standard type of masks that people wear, they come in different colours so detection becomes difficult unlike license plate detection where all license plates follow a particular convention (4) Far range detection become very difficult. So to tackle all these challenges we modeled the task of masked and unmasked face detection as a special case of object detection. We have treated the masked face and unmasked face as special objects, which we will detect in real time using popular deep learning based object detection models like YOLO, SSD and Faster R-CNN.

Related Works

Mask and Maskless Face Classification System to Detect Breach Protocols in the Operating Room

Nieto-Rodríguez et al. (2018) proposed ICDSC participants to interact with a system to classify faces into two categories: faces with and without surgical masks. The system assigns a per-personID through tracking to trigger only one alarm for a maskless face across several frames in a video. The tracking system also decreases the false positive rate. The system reaches 5 fps with several faces in VGA images on a conventional laptop. The output of the system provides confidence measures for the mask and maskless face detections, image samples of the faces, and for how many frames faces have been detected or tracked.

Methods

Overview

Recently, closed circuit television (CCTV) cameras, equipped with computer vision function by embedded systems, have been deployed in a wide range of applications, involving surveillance (Sage and Young 1999), facial recognition (Qian and Xu 2009), license plate detection of vehicles (Kashyap et al. 2018), etc. These applications require platforms that are able to sense the environment, parse scenes and react accordingly, of which the core part is scene parsing. Different surveillance applications require different stages of scene parsing, including recognizing different types of objects in the scene, locating where these objects are, and determining exact boundaries around each object. These scene parsing functions correspond to three basic research areas in the field of computer vision, namely image classification, object detection and semantic (instance) segmentation. The most common one that is adopted as a basic functional module for scene parsing in surveillance applications is visual object detection, and hence it has been the area of increasing interest. The variety of open deployment environments makes automatic scene parsing running on a surveillance platform highly demanding, which brings many new challenges to the object detection algorithms. These challenges mainly include: (1) how to deal with various variations (e.g., small sizes, shadows, illumination, angle of view, and rotation) in object's visual appearance in CCTV camera images; (2) how to deploy object detection algorithms on a surveillance platform with limited memory and computing power; (3) how to balance the detection accuracy and real-time requirements. Object detection methods based on traditional machine learning and hand-crafted features are prone to failure when dealing with these variations. One competitive approach to address these challenges is object detectors based on deep learning techniques that are popularized in recent years.

The growth of computing power (e.g., graphical processing units and dedicated deep learning chips) and the availability of large-scale labelled samples [e.g., ImageNet (Deng et al. 2009), PASCAL VOC (Everingham et al. 2015) and COCO (Lin et al. 2014)], has led to the extensive study of deep neural networks due to its fast, scalable and end-to-end learning framework. Especially, when compared with traditional shallow methods, Convolution Neural Network (CNN) (Lecun et al. May 2015) models have achieved significant improvements in image classification [e.g., DenseNet (Huang et al. 2016) and ResNet (He et al. 2015)], object detection [e.g., SSD (Liu et al. 2015) and Faster R-CNN(Ren et al. 2015)] and semantic segmentation [e.g.,UNet (Ronneberger and Fischer 2015) and Mask R-CNN He et al. 2017), etc. Since the beginning when CNN models were successfully introduced in object detection tasks R-CNN, (Girshick et al. 2013), this detection framework has attracted lots of research interest and many state-of-the-art object detectors based on CNN have been proposed in recent years. Specifically, YOLO series models (Redmon et al. 2015; Redmon and Farhadi 2016, 2018)] might be the most popular deep object detectors in practical applications as the detection accuracy and speed are well balanced. Despite that, the inference of these detectors still requires high-performance computing and a large run-time memory footprint to maintain good detection performance; it brings high computation overhead and power consumption to on-board embedded devices of surveillance platforms. Only reducing the size of the model can run the object detection method on these embedded devices. The YOLO series, SSD series and other networks have lightweight versions called YOLO-tiny (Redmon and Farhadi 2018), YOLO-Lite (Pedoeem and Huang 2018) tiny SSD (Womg et al. 2018), but the detection accuracy of the networks is greatly reduced. Therefore, how to reduce the model size and floating point operations (FLOPs) without notably reducing detection accuracy becomes an urgent problem to be solved when deploying object detectors on embedded devices in surveillance systems. The most popular method is to reduce the parameters and model size of the network by redesigning a more efficient network. For example, SqueezeNet (Iandola et al. 2016), MobileNet (Andrew 2017), ShuffleNet (Zhang et al. 2017), etc., these methods can maintain detection accuracy to a great context which significantly reduces the model size and floating point operations (FLOPs).

Moxa3K Benchmark Dataset

To train the object detectors on classes ‘mask’ and ‘nomask’, we use a dataset called ‘Moxa3K’ that we created. As the name suggests it has 3000 images, with 2800 images in the training set and 200 images in the testing set.

The dataset contains the Kaggle data set of medical masksHsun 2020, which contains 678 images. These images are mainly from China, Russia and Italy region taken during the ongoing pandemic. Most of these pictures depict a crowded area with a large number of people. As a result of which there are a lot of labels in these series (Fig. 1a). There are also 757 images which consist of close-ups of faces which include frontal faces as well as side profiles (Fig. 1b). Rest 1565 images are scraped from the internet mainly via Google image search results (Fig. 1c). These images are mainly of the ongoing COVID-19 situation in India which contains a lot of people wearing masks. All these images are evenly distributed over the train and test set. Therefore using these efficient object detection models namely YOLOv3, YOLOv3Tiny(Redmon and Farhadi 2018), SSD (Liu et al. 2015) on MoblieNetv2Sandler et al. 2018 and Faster R-CNN (Ren et al. 2015) on Inception v2 (Szegedy et al. 2016) we detected faces with masks and no masks. We trained and evaluated these models on our Moxa3K benchmark dataset and analysed their performance in this particular task in order to find the model which has a proper balance of accuracy in the challenging conditions of open deployment environments and real time compute capability on these embedded platforms of surveillance systems.

The directory structure of the dataset is shown in (Fig. 2a). All the images of the dataset are stored in the “Images” directory. The format of all the images are JPG (JPEG). The naming convention of these images are

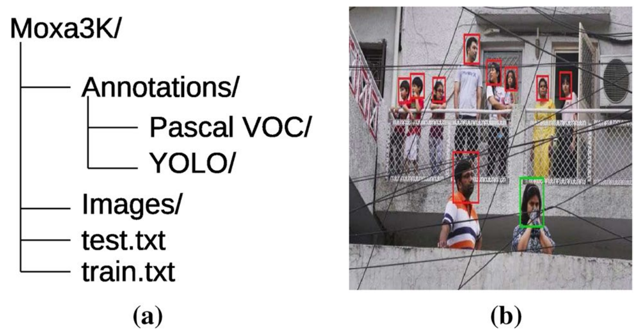


Fig. 2 **a** Directory structure of Moxa3K dataset **b** The red bounding boxes are faces without and the green ones are with masks (colour figure online)

“MOXA_<serial number of the image starting from 0 >.jpg” so the image’s names ranges from “MOXA_0.jpg” to “MOXA_2999.jpg”.

In these 3000 images of the dataset 9161 faces are without mask and 3015 faces are with masks and all of them have been labelled. These images are labelled in YOLO format which stores annotations in text files as well as in Pascal VOC(Everingham et al. 2015) format, which stores annotation XML files. The classes of these labels are ’mask’ for people wearing masks and ’nomask’ for people not wearing masks. We have used bounding boxes to localize the masked faces and unmasked faces in the images. As shown in (Fig. 2b). A tool called LabelImg (Tzutalin 2015) by Tzutalin was used in the process.

The YOLO format stores the labels in text files with the same name as each image file, for example the image file of “MOXA_101.jpg” will have the labels stored in the text file “MOXA_101.txt” containing the following information in a single line: (1) class ID, which are numbers starting from ‘0’ assigned to the classes in the dataset, in our case we have only two classes “nomask” which is assigned the ID ‘0’and “mask” which is assigned the ID of ‘1’ (2) (x,y) coordinates of the center of the object’s bounding box (3) height and width of the bounding box (absolute values of 2, 3 are divided by the image height and width and stored). (Figure 3) shows the format of YOLO labels. All the text files ranging from “MOXA_0.txt” to “MOXA_2999.txt” containing the corresponding labels are stored in the “YOLO” directory inside the “Annotations” directory. Figure 4b shows

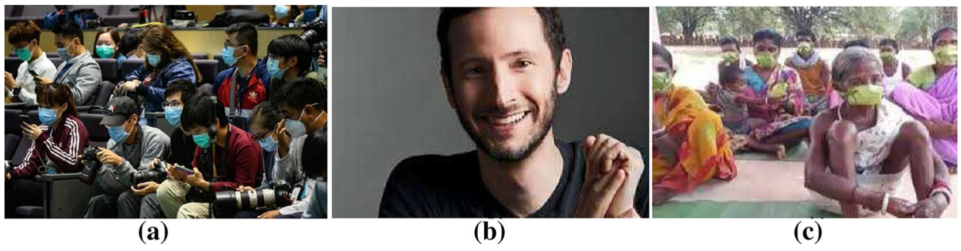
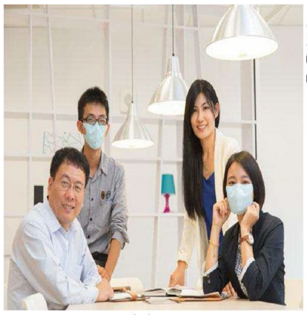


Fig. 1 **a** Image from China region **b** Closeup image of a face **c** Image from India

```
<object-class> <x> <y> <width> <height>
```

Fig. 3 YOLO label format



```
1 0.299194 0.367512 0.117742 0.214286
1 0.799194 0.594470 0.159677 0.262673
0 0.217742 0.595622 0.106452 0.205069
0 0.668548 0.355991 0.101613 0.186636
```

(a) (b)

Fig. 4 Image with corresponding YOLO labels

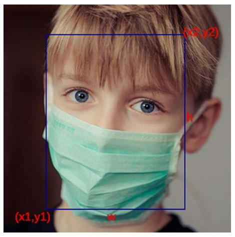
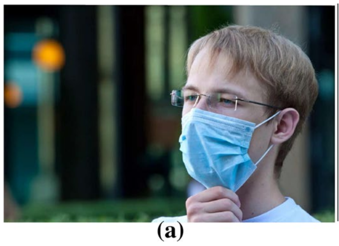


Fig. 5 Coordinates of the bounding boxes in VOC format

annotation of Fig. 4a from our dataset, in the annotation we can see first two lines starting with ‘1’ which corresponds to 2 masked faces in Fig. 4b, and the remaining two lines start with ‘0’ which corresponds to the two faces without masks in the image.

The Pascal VOC (Everingham et al. 2015) format stores details of the objects in images in an XML file. These XML files in our dataset are of the same name as the corresponding images, for example the image “MOXA_49.jpg” will have all the labels stored in the “MOXA_49.xml” file. These XML files contain informations like: (1) name of the image inside<filename> tag (2) name of the dataset in the<database> tag which in our case will be Moxa3k (3) dimensions of the image in<size> tag (4) the details of the object are stored in the<object> tag, inside the<object> tag there is<name> tag which stores the name of the objects, in our case these names will be “mask” and “nomask”, the coordinates of the bounding boxes in <bdnbox> tag in format:<bdnbox><xmin></xmin<ymin></ymin><xmax></xmax><ymax></ymax></bdnbox>, here the<xmin>,<ymin> tags store (x1,y1) coordinates of the (Fig. 5) respectively and <xmax> ,<ymax> tags store (x2,y2) coordinates of the (Fig. 5). Figure6b show annotation of Fig. 6a from our dataset, in the annotation we can see a single <object> tag with <name> “mask” and the



```
-<annotation>
  <folder/>
  <filename>MOXA_49.jpg</filename>
  <path>../../dataset/moxa3K/train//MOXA_49.jpg</path>
 -<source>
   <database>Moxa3k</database>
  </source>
 -<size>
   <width>1688</width>
   <height>1125</height>
   <depth>3</depth>
  </size>
  <segmented>0</segmented>
 -<object>
   <name>mask</name>
   <pose>Unspecified</pose>
   <truncated>0</truncated>
   <difficult>0</difficult>
  -<bndbox>
     <xmin>841</xmin>
     <ymin>213</ymin>
     <xmax>1422</xmax>
     <ymax>969</ymax>
   </bndbox>
  </object>
</annotation>
```

(a) (b)

Fig. 6 Image with corresponding VOC labels

coordinates of the bounding boxes. All the XML files are stored in “Pascal VOC” directory.

In our dataset we have also taken care of the boundary conditions, such as people wearing handkerchief as mask, and people covering their face with their hands or with a cloth in their hand as shown in Fig. 2b, the green bounding box contains a lady covering her mouth with a handkerchief which is labelled as ‘mask’ . We also have several samples where there are “mask” and “no mask” faces which are blurred like in Fig. 7a to increase the robustness of the object detections. In addition there are samples which have a top angle view to replicate the viewing angle of CCTV cameras (Fig. 7b), so that we can match the deployment environments of the models which will be trained using this dataset. Besides this we also have samples of images where the faces are rotated at particular angles (Fig. 7c) to facilitate rotation invariant detections. Our dataset also includes samples with both frontal face (Fig. 1b) as well as side profiles of faces (Fig. 6a) and it contains samples of crowded areas (Fig. 7d) with lots of people in a single frame where the size of the bounding boxes of some faces are very small and also containing samples with only one person in the frame where the bounding box is fairly large. We also have images with different illumination condition, and also with different weather conditions like winter (Fig. 7e) where foggy environments make object detections very difficult, and also there are samples taken in sunny summer environments (Fig. 7f). All these factors greatly increase the robustness of the object detectors trained on this dataset.

Experiments and Performance on Various Models

Here we have particularly worked with single shot object detectors like YOLOv3 (Redmon and Farhadi 2018),YOLOv3Tiny (Redmon and Farhadi 2018), SSD (Liu et al. 2015), and double shot detectors like Faster R-CNN (Ren et al. 2015), which were trained on our dataset, and their performances were analyzed. In the case of SSD and Faster R-CNN we went with efficient and lightweight networks like MobileNet v2 (Sandler et al. 2018) and Inception v2 (Szegedy et al. 2016) respectively to improve performances at low compute capabilities without affecting

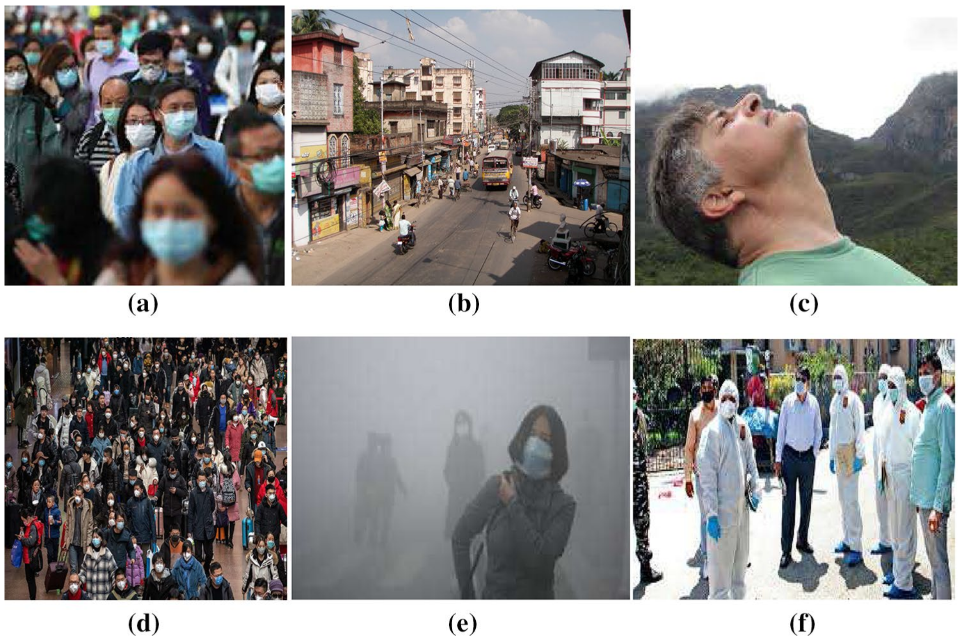


Fig. 7 **a** Blurred faces **b** top angle camera view **c** rotated face **d** crowded environments **e** foggy environments **f** sunny environments

accuracy of detection for real time inferences. For YOLO models: YOLOv3 and YOLOv3Tiny we used the native Darknet network. The main task here is to find the object detector that balances both accuracy and compute speed, as the main use of this system is on edge devices of the surveillance platforms where real time detections are very important.

Hardware and Enviornments

All the models are trained on an instance of Google Colab platform with an Intel(R) Xeon(R) CPU @ 2.30GHz processor, 13 GB RAM and 1 Nvidia Tesla P-100 GPU running Linux OS (Ubuntu 15.04 LTS). The inference tests and evaluation of all these models are done on a machine with Intel Core i5-9300H (9th Gen) @2.40 GHz processor, with 8 GB of RAM and 1 Nvidia GTX 1650 GPU running Linux OS (Ubuntu 20.04 LTS). All measurements of images and frame sizes are taken in pixels. We report all inference time in ms per 1280x720 image including all pre and post-processing and number of frames of 1280x720 processed per second (FPS).

YOLOv3

YOLOv3 is one of the most popular object detectors which tackles the task of real time detection with its state of art algorithms. Prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections. YOLO uses a totally different approach. It applies a single neural network to the entire image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities. YOLO has several advantages over classifier-based systems. It looks at the whole image at test time so its predictions are informed by global context in the image.

YOLOv3 makes an incremental improvement to the YOLO series models in object detection accuracy. First, YOLOv3 adopts a new backbone network, i.e., Darknet-53, as a feature extractor. Darknet-53 uses more successive 3×3 and 1×1 convolutional layers rather than Darknet-19 in YOLOv2 and organizes them as residual blocks (Redmon and Farhadi 2018). Hence, Darknet-53 is much more powerful than Darknet-19 but still more efficient than ResNet-101 (He et al. 2015). Second, YOLOv3 predicts bounding boxes at three different scales by following the idea of feature pyramid network for object detection (Redmon and Farhadi 2018). Three detection headers separately built on the top of three feature maps with different scales are responsible for detecting objects with different sizes. Each grid in the detection header is assigned with three different anchors, and thus predicts three detections that consist of 4 bounding box offsets, 1 objectiveness and C class predictions. The final result tensor of the detection header has a shape of $N \times N \times (3 \times (4 + 1 + C))$, where N×N denotes the spatial size of the last convolutional feature map.

Training Following the default configurations in Darknet (Redmon and Farhadi 2018), we trained YOLOv3 on Moxa3K benchmark dataset using SGD with the momentum of 0.9 and weight decay of 0.0005. We use an initial learning rate of 0.001 that is decayed by a factor of 10 at the iteration step of 3200 and 3600. We set the maximum training iteration as 4000 and use mini-batch size of 64. We set the size of input image as 416 for YOLOv3. Multiscale training is enabled by randomly rescaling the sizes of input images. We initialized the backbone networks of the model with the weights pre-trained on ImageNet (Deng et al. 2009).

Evaluation and analysis With the trained weights we ran evaluations on test set of Moxa3K benchmark dataset, in native Darknet environment. For hyperparameters we used the same configuration as the training with input image size as 416. The Inference time of the model is 23 FPS,with total binary floating point operations (BFLOPS) 65.312, 65.2M parameters and a model size of 246.3MB. The evaluation was done on the following parameters: (1) precision, (2) recall, (3) mean of average precision (mAP) measured at 0.5 intersection over union (IoU), (4) F1-score for each class and the following results were obtained (Table.1).

We also ran evaluations by varying the input image size on the following parameters: (1) precision, (2) recall, (3) mean of average precision (mAP) measured at 0.5 intersection over union (IOU), (4) F1-scores (5) FLOPs and (6) inference time as frames per second (FPS)and these are the results obtained (Table 2).

We can see that the mAP of 608 × 608 input size is 3% greater than the 416 × 416 input size model, however its inference time is 94.77% greater than the later. Another important observation is when we increase the input size to 832 × 832 the mAP decreases by 3.5% with an increase in the inference time by 207.68% from the 416 × 416 model. So we can say that the YOLOv3 is not that sensitive to input size and the 416 × 416 model is most suitable for real time inference.

YOLOv3 Tiny

Similar to YOLOv3 (Redmon and Farhadi 2018) the creators also designed a smaller model for constrained environments which is the YOLOv3 Tiny. The YOLOv3 Tiny runs on the same algorithm but uses a lighter version of the model with only 2 YOLO layers. Being light weight it is able to achieve much faster inference with a low processing overhead. Which is very essential for deployment in surveillance platforms.

Table 2 Performance analysis over various input sizes of YOLOv3

	Input size		
	416 x416	608x608	832x832
Precision	0.76	0.76	0.73
Recall	0.79	0.80	0.75
F1-Score	0.77	0.78	0.74
mAP@50	63.99%	66.84%	61.73%
BFLOPS	65.31	139.512	261.247
FPS	21.2	10.9	6.9
Inference time (ms)	47.1	91.74	144.92

Training Following the default configurations in Darknet (Redmon and Farhadi 2018), we trained YOLOv3 (Redmon and Farhadi 2018) Tiny on Moxa3K benchmark dataset using SGD with the momentum of 0.9 and weight decay of 0.0005. We use an initial learning rate of 0.001 that is decayed by a factor of 10 at the iteration step of 3200 and 3600. We set the maximum training iteration as 4000 and use a mini-batch size of 64. We set the size of the input image as 416 for YOLOv3 Tiny. Multiscale training is enabled by randomly rescaling the sizes of input images. We initialized the backbone networks of the model with the weights pre-trained on ImageNet (Deng et al. 2009).

Evaluation and analysis With the trained weights we ran evaluations on test set of Moxa3K benchmark dataset, in native Darknet environment. For hyperparameters we used the same configuration as the training with input image size as 416. The Inference time of the model is 138 FPS with total binary floating point operations (BFLOPS) 5.449, 8.7M parameters and a model size of 34.7MB. The evaluation was done on the following parameters: (1) precision, (2) recall, (3) mean of average precision (mAP) measured at 0.5 intersection over union (IoU), (4) F1-score for each class and following results were obtained (Table 3).

Like Yolov3 we also ran evaluations by varying the input image size on the following parameters: (1) precision, (2)

Table 1 Class wise performance analysis of YOLOv3

	Class		
	Nomask	Mask	Overall
Images	200	200	200
Instances	135	875	1015
Precision	0.45	0.81	0.76
Recall	0.25	0.77	0.79
F1-Score	0.32	0.78	0.77
mAP@50	46.69%	81.26%	63.99%

Here instances are no. of objects detected

Table 3 Class wise performance analysis of YOLOv3 Tiny

	Class		
	Nomask	Mask	Overall
Images	200	200	200
Instances	112	713	825
Precision	0.38	0.76	0.71
Recall	0.12	0.64	0.66
F1-Score	0.18	0.69	0.69
mAP@50	41.06%	71.48%	56.27%

Instances are number of objects detected

Table 4 Performance analysis over various input sizes of YOLOv3Tiny

	Input size		
	416 × 416	608 × 608	832 × 832
Precision	0.71	0.72	0.68
Recall	0.66	0.70	0.68
F1-Score	0.77	0.78	0.74
mAP@50	56.27%	55.08%	56.57%
BFLOPS	5.449	11.640	21.797
FPS	138	72	46.5
Inference time (ms)	7.2	13.8	21.5

recall, (3) mean of average precision (mAP) measured at 0.5 intersection over union (IOU), (4) F1-scores (5) FLOPs and (6) inference time as (FPS) and these are the results obtained (Table 4). Here the 416 × 416 model wins by a large margin and is well suited for real time applications.

SSD: Single Shot Multibox Detection on Mobilenet v2

Single Shot object detection or SSD (Liu et al. 2015) takes one single shot to detect multiple objects within the image. SSD is faster than R-CNN because in R-CNN (Girshick et al. 2013) we need two shots, one for generating region proposals and the other for detecting objects. SSD is used to speed up the process by eliminating the region proposal network which results in a drop in accuracy, so we combined the MobileNet v2 (Sandler et al. 2018) and SSD to get better accuracy. Thus it performs fairly well in constrained environments especially like in our case which requires real time performance with low computation power. Resnet (He et al. 2015),VGG (Simonyan and Zisserman 2014) or alexnet (Krizhevsky et al. 2012) has a large network size and it increases the number of computations whereas in Mobilenet v2 there is a simple architecture which reduces the processing overhead without hampering the detection to a great extent. Hence, we used mobilenet v2 over Resnet or VGG to facilitate real time performances with low processing overhead.

Training We trained SSD with Mobilenet v2 backbone using tensorflow object detection API(Huang et al. 2016) with tensorflow 1.15 on Moxa3K benchmark dataset using SGD with the momentum of 0.9 and weight decay of 0.00004. We use an initial learning rate of 0.004 that is decayed by a factor of 0.95 with decay steps of 800720. We set the maximum training steps as 4000 and use a batch size of 24. We used the size of input image as 300 for Mobilenet v2 SSD. We initialized the backbone network of the model with the weights pre-trained on MS-COCO (Lin et al. 2014).

Evaluation and analysis With the trained weights we ran evaluations on test set of Moxa3K benchmark dataset, on tensorflow 1.15 using tensorflow object detection API (Huang et al. 2016) evaluation scripts with COCO metrics (Lin et al. 2014). For hyperparameters we used the same configuration as the training with input image size as 300. The Inference time of the model is 67.18 FPS [this inference was ran using the OpenCV DNN Modules backed with CUDA support (Samaga 2018)], and a model size of 19.3 MB . The evaluation yielded the following results (Table 5).

From the (Table 5) it is evident that the model is more efficient in detecting masks over no mask. With an inference time of 14.88 ms it is well capable of performing realtime detections however the quality of detections comes at cost as we have relatively lower mAP.

Faster R-CNN on Inception v2

Traditional CNN (Lecun et al. 2015) divides the image into multiple regions and then classifies each region into various classes. Hence it needs a lot of regions to predict accurately and therefore high computation time. To solve this problem Girshick et al proposed R-CNN (Girshick et al. 2013) which uses selective search to generate regions. It extracts around 2000 regions from each image and for each region, CNN is used to extract specific features. Finally, these features are then used to detectobjects. Due to this it takes high computation time as each region is passed to the CNN separately. Also, it uses three different models for making predictions. To solve this problem Girshick had suggested Fast R-CNN (Girshick 2015), which passes the entire image to ConvNet which generates regions of interest (instead of passing the extracted regions from the image). Also, instead of using three different models (as we saw in R-CNN), it uses a single model which extracts features from the regions, classifies them into different classes, and returns the bounding boxes. All these steps are done simultaneously, thus making it execute faster as compared to R-CNN. Fast R-CNN is, however, not fast enough when applied on a large dataset as it also uses selective search for extracting the regions. Finally Shaoqing Ren et al proposed Faster R-CNN (Ren et al. 2015), which fixes the problem of selective search by replacing it with Region Proposal Network (RPN). We first extract feature maps from the input image using Convolution Neural Network and then pass those maps through a RPN which returns object proposals. Finally, these maps are classified and the bounding boxes are predicted. Despite being significantly faster than its predecessors it still has a huge computational overhead. Although it is very

Table 5 Class wise performance analysis of SSD

Class	Images	mAP@50
Nomask	200	29.81%
Mask	200	63.24%
Overall	200	46.52%

accurate it might not be well suited for the deployment environments of our task which requires real time performances at low computational overhead.

Training We trained Faster R-CNN (Ren et al. 2015) with Inception v2 (Szegedy et al. 2016) backbone using tensorflow object detection API (Huang et al. 2016) with tensorflow 1.15 on Moxa3K benchmark dataset using SGD with the momentum of 0.9 and weight decay of 0.00004. We use an initial learning rate of 0.004 that is decayed by a factor of 0.95 with decay steps of 800720. We set the maximum training steps as 4000 and use batch size of 24. We used the size of input image as 300 for Inception v2 Faster R-CNN. We initialized the backbone network of the model with the weights pre-trained on MS-COCO (Lin et al. 2014).

Evaluation and analysis With the trained weights we ran evaluations on test set of Moxa3K benchmark dataset, on tensorflow 1.15 using tensorflow object detection API (Huang et al. 2016) evaluation scripts with COCO metrics (Lin et al. 2014) (Table6). For hyperparameters we used the same configuration as the training with input image size as 300. The Inference time of the model is 14.88 FPS (Qian and Xu 2009) [this inference was ran using the OpenCV DNN Modules backed with CUDA support (Samaga 2018)], a model size of 52.3MB. The evaluation yielded the following results (the results are yielded with maximum number of detections as 100).

We see that the model has a mAP of 60.5% with an IoU of 50%, and with a higher IoU of 75% it's mAP increases to 89% which tells us that model performs really well with higher levels of threshold values.

Results and Comparison

Comparing the performances of the stated models on Moxa3K benchmark dataset (Table 7) we see that the maximum mAP is obtained with YOLOv3 608x608, however with the increased input size and the comparatively heavy processing overhead it fails to give real time inference.

Table 6 Performance analysis of faster R-CNN

Metric	IoU	Area	Result
	50 to 95%	All	22.8%
	50%	All	60.5%
Average precision	75%	All	89.0%
	50 to 95%	Small	18.9%
	50 to 95%	Medium	32.7%
	50to 95%	Large	29.5%
	50 to 95%	All	42.0%
Average recall	50 to 95%	Small	38.8%
	50 to 95%	Medium	49.2%
	50 to 95%	Large	50.0%

Table 7 mAP and FPS of trained models on Moxa3K dataset

Model	mAP@50	FPS
YOLOv3 414x414	63.99	21.2
YOLOv3 608x608	66.84	10.9
YOLOv3 832x832	61.73	6.9
YOLOv3Tiny 414x414	56.27	138
YOLOv3Tiny 608x608	55.08	72
YOLOv3Tiny 832x832	56.57	46.5
SSD 300 MobliNetv2	46.52	67.1
F-RCNN 300 Inceptionv2	60.5	14.8

However YOLOv3 with 416 x 416 gives a FPS of 21.2 with an mAP of 63.99%. But considering the fact that the main application of this detection system is to be deployed in a surveillance based platform supported by embedded systems with limited compute capability the YOLOv3 416x416 model will fail to produce real time inference. The YOLOv3 Tiny model however fits perfectly for the use case, it has a satisfactory mAP of 56.27% with a FPS of 138. It provides a good balance of accuracy and real time inference. With a very less processing overhead it is able to provide real time detections of masked and no masked faces on the surveillance platforms with limited compute power.We ran some test samples on the trained models to compare their performances (Fig. 8).

Conclusion

The idea of detecting whether people wearing face masks or not find its best implementation in areas with very high footfall, such as markets, offices, rail stations etc. because the chances of transmission are highest in these areas. Our system can be implemented at any location which can provide a video input, for eg: live feed, recorded video feed, etc. Thus a detection model which has the ability to run at real time and is accurate enough to detect small objects like masked faces, can be very effective in these edge applications in surveillance platforms. In future to improve the detection system more advanced object detection models can be used. In addition to that the dataset can be expanded further to enhance the training and evaluation process so that a better analysis of performance of the various object detection models can be done on it.

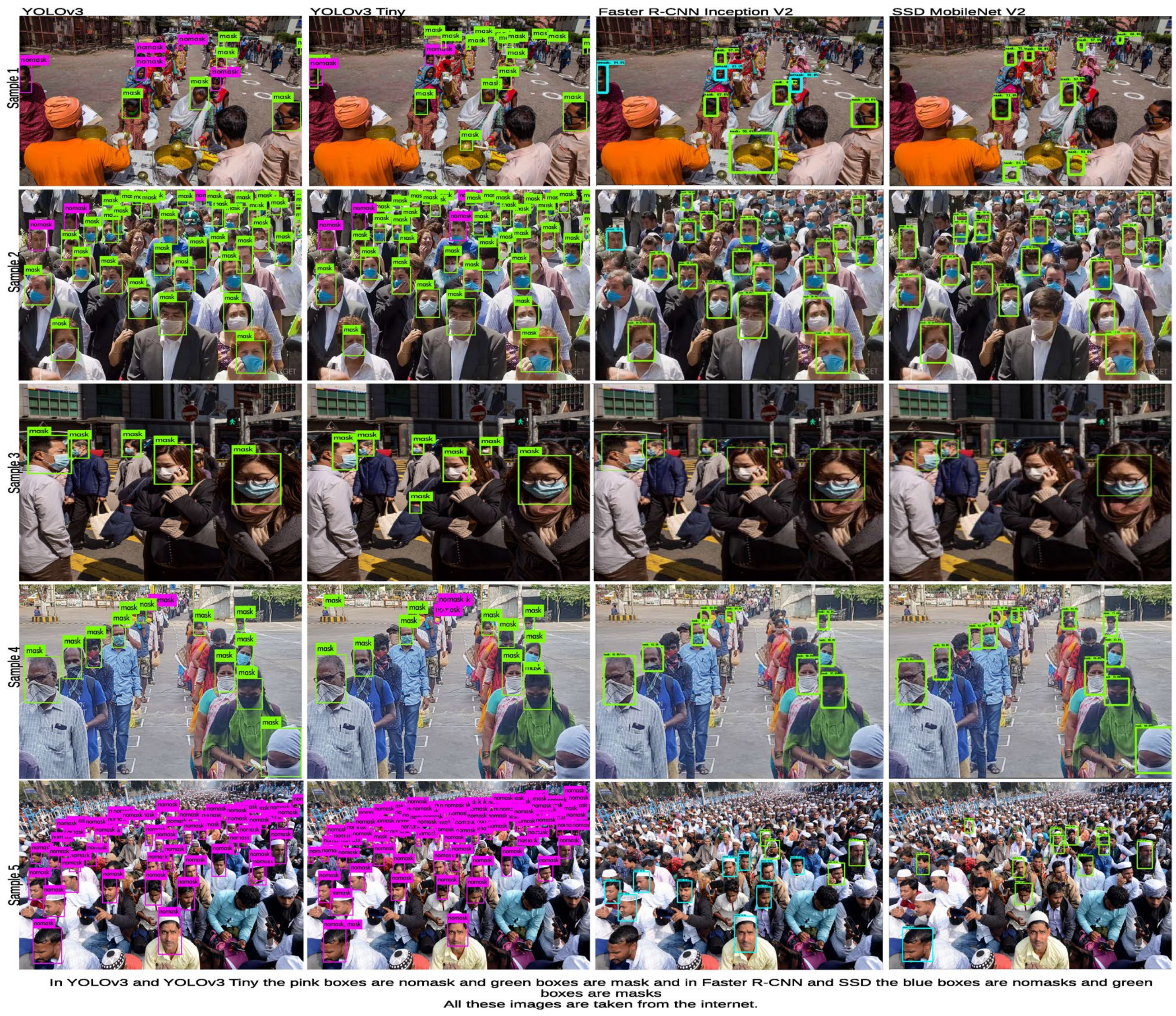

YOLOv3
YOLOv3 Tiny
Faster R-CNN Inception V2
SSD MobileNet V2
Sample 1
Sample 2
Sample 3
Sample 4
Sample 5
In YOLOv3 and YOLOv3 Tiny the pink boxes are nomask and green boxes are mask and in Faster R-CNN and SSD the blue boxes are nomasks and green boxes are masks
All these images are taken from the internet.

Fig. 8 Comparison of the trained models on some test samples

Acknowledgements This project was supported and guided by Prof. Rajib Bandopadhyay (Dept. Instrumentation and Electronics Engineering, Jadavpur University). Mainak Pal (Dept. Electronics and Telecommunication Engineering) and Pritthijit Nath (Dept. Computer Science Engineering) of Jadavpur University also provided their support.

Data Availibility Statement All the config files and weights of all the trained object detection models along with our dataset Moxa3K has been provided here: https://shitty-bots-inc.github.io/MOXA/index.html, https://MOXA/index.html.

Compliance with ethical standards

Conflict of interest The authors declare that they do not have any conflict of interest.

Availability of code All codes are available here:https://shitty-bots-inc.github.io/MOXA/index.html link.

References

Deng J, Dong W, Socher R, Li L, Li K, Fei-Fei L (2009) Imagenet: a large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition, pp 248–255

Everingham M, Eslami SMA, Van Gool L, Williams CKI, Winn J, Zisserman A (2015) The pascal visual object classes challenge: a retrospective. Int J Comput Vis 111(1):98–136

Girshick R (2015) Fast r-cnn

Girshick R, Donahue J, Darrell T, Malik J (2013) Rich feature hierarchies for accurate object detection and semantic segmentation

He K, Zhang X, Ren S, Sun J (2015) Deep residual learning for image recognition

He K, Gkioxari G, Dollár P, Girshick R (2017) Mask r-cnn

Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H (2017) Efficient convolutional neural networks for mobile vision applications, Mobilenets

Hsun TC (2020) Medical masks dataset, 02

Huang G, Liu Z, van der Maaten L, Weinberger KQ (2016) Densely connected convolutional networks

Huang J, Rathod V, Sun C, Zhu M, Korattikara A, Fathi A, Fischer I, Wojna Z, Song Y, Guadarrama S, Murphy K (2016) Speed/accuracy trade-offs for modern convolutional object detectors

Iandola FN, Han S, Moskewicz MW, Ashraf K, Dally WJ, Keutzer K (2016) Squeezenet: Alexnet-level accuracy with 50x fewer parameters and<0.5mb model size

Kashyap A, Suresh B, Patil A, Sharma S, Jaiswal A (2018) Automatic number plate recognition. In: 2018 international conference on advances in computing, communication control and networking (ICACCCN), pp 838–843

Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Proceedings of the 25th international conference on neural information processing systems - volume 1, NIPS'12, page 1097–1105, Red Hook, NY, USA, Curran Associates Inc

Lecun Y, Bengio Y, Hinton G (2015) Deep learning. Nat Cell Biol 521(7553):436–444

Lin T-Y, Maire M, Belongie S, Bourdev L, Girshick R, Hays J, Perona P, Ramanan D, Lawrence Zitnick C, Dollár P (2014) Microsoft coco: Common objects in context

Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu C-Y, Berg AC (2015) Ssd: Single shot multibox detector

Nieto-Rodríguez, Mucientes M, Brea VM (2018) Mask and maskless face classification system to detect breach protocols in the operating room. In: Proceedings of the 9th international conference on distributed smart cameras, ICDSC '15, page 207–208, Association for Computing Machinery, New York, NY, USA

Pedoeem J, Huang R (2018) Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers

Qian Z, Xu D (2009) Research advances in face recognition. In: 2009 Chinese conference on pattern recognition, pp 1–5

Redmon J, Farhadi A (2016) Yolo9000: better, faster, stronger

Redmon J, Farhadi A (2018) Yolov3: an incremental improvement. arXiv

Redmon J, Santosh D, Ross G, Farhadi A (2015) Unified, real-time object detection, you only look once

Ren S, He K, Girshick R, Jian S (2015) Towards real-time object detection with region proposal networks, Faster r-cnn

Ronneberger O, Fischer P, Thomas B (2015) Convolutional networks for biomedical image segmentation, U-net

Sage K, Young S (1999) Security applications of computer vision. IEEE Aerosp Electron Syst Mag 14(4):19–29

Samaga YB (2018) Opencv with cuda support. https://github.com/YashasSamaga/opencv

Sandler M, Howard A, Zhu M, Zhmoginov A, Chen L-C (2018) Mobilenetv2: inverted residuals and linear bottlenecks

Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition

Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z (2016) Rethinking the inception architecture for computer vision. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, pp 2818–2826. https://doi.org/10.1109/CVPR.2016.308

Tzutalin. Labelimg. https://github.com/tzutalin/labelImg, 2015

Womg A, Shafiee MJ, Li F, Chwyl B (2018) Tiny SSD: a tiny single-shot detection deep convolutional neural network for real-time embedded object detection. In: 2018 15th Conference on Computer and Robot Vision (CRV), Toronto, ON, pp 95–101. https://doi.org/10.1109/CRV.2018.00023

Xiangyu Z, Xinyu Z, Mengxiao L, Sun J (2017) An extremely efficient convolutional neural network for mobile devices, Shufflenet