Chapter 2

Literature Review

We researched a number of relevant papers in order to choose a method for face mask detection and social distancing detection which can be implemented on the Jetson Nano board for real-time implementation. We concentrated on publications that compared several types of existing algorithms, as well as the benefits and drawbacks of each approach.

Table 1 Comparison of various Deep Learning techniques for Face Mask and Social Distancing Detection

Algorithm	Methodology	Results
Facial Mask Detection using CNN	Preprocessing steps includes filtering images, resizing images, denoising images and then applying the CNN architecture.	98.7%
Face mask detection using ResNet50, MobileNetV2 and AlexNet	A pretrained model ResNet50 is used and in order to achieve trade-off between accuracy and computational time, an image complexity predictor is proposed.	ResNet50: 945 MobileNetV2: 83.2% Alexnet: 75%
Real time face mask and social distancing detection using YOLOv4	YOLOv4 is used for both facemask and social distancing detection. Dense blocks are used which contains Batch Normalization and ReLU followed by CNN.	94.75%
Face mask detection using KNN, SVM and MobileNet	Data augmentation is doneto increase datasets by rotating, flipping and bluring the images. The next step is to apply MobileNetV2 followed by Adam optimizer to prevent over-fitting.	SVM: 89.4% KNN: 87.85% MobileNet: 94.2%

DNN based face mask detection using single shot multibox detector and MobileNetV2	Proposed a model named SSDMNV2 using OpenCV, DNN, Tensorflow, Keras and MobileNetV2	Accuracy: 92.64% F1 Score: 0.93
Comparison of different algorithms like hybrid deep transfer learning, YOLOv2, Inceptionv3, R-CNN,MobileNetV2	The study adopted a literature review to achieve primary goal of identifying the models which detects face masks.	Hybrid model: 99.64% Inceptionv3: 99.9% R-CNN: 68.72% YOLOv2: 81% MobileNetV2: 93%
Social Distancing Detection using YOLOv2	Takes the thermal input image and applies Object Detection by filtering only people class. Checks the number of persons and the distances between them and then decide if it is safe or not	Accuracy: 95.6% Precision: 95% Recall:96%
Social distancing with person detection using YOLOv3 and Deepsort techniques	YOLO v3 model is used to separate humans from the background and Deepsort approach to track the identified people with the help of bounding. Achieves best results with balanced mAP and FPS score to monitor the social distancing in real-time.	YOLOv2: mAP = 8.46, FPS = 23 SSD: mAP = 0.691, FPS=10 Faster R-CNN: mAP = 0.969, FPS = 3

2.1 MobileNetV2 for Face mask detection

MobileNetV2 is a Google-developed approach based on Convolutional Neural Networks (CNN), which has enhanced performance and efficiency. MobileNetV2 is a powerful feature extractor for detecting and segmenting objects. MobileNetV2 is a 53-layer deep convolutional neural network. The ImageNet database can be used to load a pretrained version of the network that has been trained on over a million photos. The network can classify photos into 1000 different object categories, including keyboards, mouse, pencils, and a variety of animals. As a result, the network has learned a variety of rich feature representations of images. The network's picture input size is 224 × 224 pixels. MobileNetV2 expands on the concepts of MobileNetV1, i.e. utilising depthwise separable convolution as a cost-effective building element which is a form of factorized convolutions which factorize a standard convolution into a depthwise convolution and a 1×1 convolution called a pointwise convolution. The depthwise convolution used by MobileNets applies a single filter to each input channel. The pointwise convolution then applies a 1×1 convolution to combine the outputs the depthwise convolution. The depthwise separable convolution divides this into two layers: one for filtering and the other for combining. This factorization reduces computing time and model size significantly. However, V2 adds two new architectural features: 1) linear bottlenecks between layers, and 2) shortcut connections between bottlenecks.

2.2 YOLOv3 for Social Distancing Detection

2.2.1 Why YOLO?

There are 3 primary object detection algorithms namely R-CNN and its variants, Single Shot Detector (SSDs) and YOLO. The R-CNN family of networks give accurate results but the disadvantage is their speed. Their performance is slow, obtaining only 5FPS on a GPU. SSDs and YOLO use a one-stage detector strategy and tend to be less accurate than two-stage detectors but are significantly faster. The YOLO v2 can process images at 40–90 FPS while YOLO v3 allows us to easily tradeoff between speed and accuracy, just by changing the model size without any retraining. YOLO v4/v5 which are improvements take a lot of computing and training.

2.2.2 YOLOv3 Architecture

YOLO is a Convolutional Neural Network (CNN) for performing object detection in real-time. YOLOv3 uses a variant of Darknet, a framework to train neural networks, which originally has 53 layers. For the detection task another 53 layers are stacked onto it, accumulating to a total of a 106-layer fully convolutional architecture. This explains the reduction in speed in comparison with the YOLOv2 which only has 30 layers. YOLO and other convolutional neural network algorithms "score" regions based on their similarities to predefined classes. High-scoring regions are noted as positive detections of whatever class they most closely identify with. For example, in a live feed of traffic, YOLO can be used to detect different kinds of vehicles depending on which regions of the video score highly in comparison to predefined classes of vehicles.

Chapter 3

Hardware and Software Design

We'll look at the hardware and software design of our model, as well as the process we used to train it and improve its results, in this chapter. The hardware design section includes the steps to be carried out in Jetson Nano board and the software design section covers the implementation of two different algorithms, i.e. MobileNetV2 and YOLOv3.

3.1 Hardware Design

The hardware for this project is planned to be built on the Jetson Nano board by setting up a camera module to capture real-time photos and make predictions based on those images. The Jetson Nano board is set up by installing the operating system on a micro-SD card.

Steps to be followed for porting an OD on Jetson Nano:

- Download the Jetson Nano Developer Kit SD Card Image
- Format your microSD card using SD Memory Card Formatter from the SD Association.
- Use Etcher to write the Jetson Nano Developer Kit SD Card Image to your microSD card
- Click on “Select image” in Etcher and choose the zipped image file downloaded earlier.
- Click “Select drive” and choose the correct device
- Click “Flash!” and it will take Etcher about 10 minutes to write and validate the image if your microSD card is connected via USB3.

The SD card can then be inserted on Jetson Nano. Setting up the Jetson Nano Board to establish a compatible environment by installing the operating system and making the hardware ready is all that is required for hardware design.

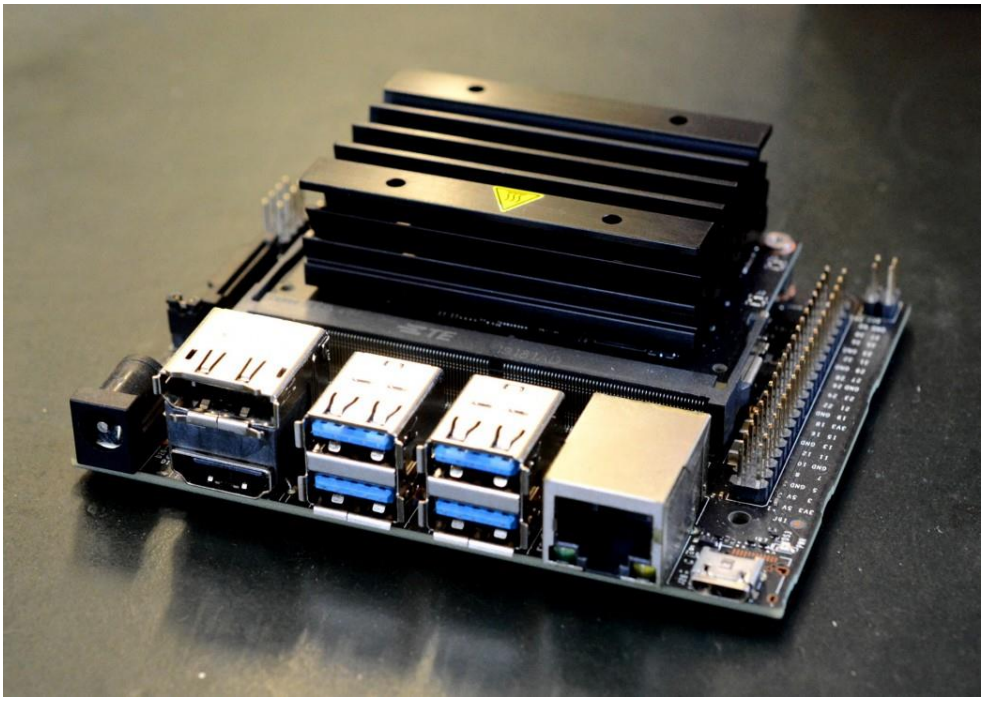

Figure 2 Jetson Nano Board

3.2 Software Design

3.2.1 Methodology for Face Mask Detection

There are 3 layers of MobileNetV2:

- The first layer is 1×1 convolution with ReLU6.
- The second layer is the depthwise convolution.
- The third layer is another 1×1 convolution but without any non-linearity.

Data Preprocessing

The dataset for the model has 3830 images divided into 2 categories: with mask and without mask. The model is trained for 20 Epochs with a batch size of 32. The learning rate is 10^{-4}. Pre-processing steps include converting images to arrays and performing one-hot encoding on the labels. The target size of the image is set to 224 x 224 pixels. The model is split using train_test_split with test_size of 0.2.

```
# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

data = np.array(data, dtype="float32")
labels = np.array(labels)

(trainX, testX, trainY, testY) = train_test_split(data, labels,
  test_size=0.20, stratify=labels, random_state=42)
```

Model Training

ImageDataGenerator is used for creating multiple copies of a single image with different angle rotation, zoom range, width shift range and height shift range. MobileNetV2 network is used which performs computations faster than a Convolutional Neural Network. It uses less parameters but also has a disadvantage of obtaining less accurate results. Imagenet is a pretrained model used for a dataset with images.

```
# construct the training image generator for data augmentation
aug = ImageDataGenerator(
  rotation_range=20,
  zoom_range=0.15,
  width_shift_range=0.2,
  height_shift_range=0.2,
  shear_range=0.15,
  horizontal_flip=True,
  fill_mode="nearest")

# load the MobileNetV2 network, ensuring the head FC layer sets are
# left off
baseModel = MobileNetV2(weights="imagenet", include_top=False,
  input_tensor=Input(shape=(224, 224, 3)))
```

The next step is average pooling and the pool size is set to (7,7). Firstly, the layers are flattened and then a dense layer with 128 neurons is added. The model is trained for 2 layers with ReLu and Softmax activation function and Adam optimiser is used. Using matplotlib, training loss, confusion matrix and accuracy is calculated.

Real-time Model with Camera

Using the cv2.dnn module of OpenCV, face detection can be performed. The model trained for mask detection is imported along with this. The OpenCV is paired with the Caffemodel for face

detection and we use the pretrained weights model which is popularly used viz. res10_300x300_ssd_iter_140000.caffemodel. A dialog box named frame appears on the screen which takes the input from the webcam and reads frames one by one. We pass each frame to a detection and prediction function with other arguments such as the facenet and masknet models. The return values are tuples of location of the face and mask as well as the prediction text which displays if the mask is present or not. Rectangle is drawn for face detection using coordinates of the X and Y axis. We were able to find exceptional results in terms of accuracy and precision of prediction as compared to the literature survey results.

3.2.2 Methodology for Social Distancing Detection

The YOLOv3 is pre trained using the coco dataset which is a large-scale object detection, segmentation, and captioning dataset. The dataset has 330K images, 1.5 million object instances, 80 object categories, 91 stuff categories, 5 captions per image and 250,000 people with key points. In our model, we set the classID to 'person' so that it detects different people inside the test frame. The algorithm first separates an image into a grid. Each grid cell predicts some number of boundary boxes around objects that score highly with the aforementioned predefined classes. Each boundary box has a respective confidence score of how accurate it assumes that prediction should be and Non-Maxima Suppression ensures that we have only one object per bounding box.

```
def detect_people(frame, net, ln, personIdx=0):
    # grab the dimensions of the frame and  initialize the list of results
    (H, W) = frame.shape[:2]
    results = []

    # construct a blob from the input frame and then perform a forward
    # pass of the YOLO object detector, giving us our bounding boxes and associated probabilities
    blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416), swapRB=True, crop=False)
    net.setInput(blob)
    layerOutputs = net.forward(ln)

    # initialize our lists of detected bounding boxes, centroids, and confidences, respectively
    boxes = []
    centroids = []
    confidences = []

    # loop over each of the layer outputs
    for output in layerOutputs:
        # loop over each of the detections
        for detection in output:
            scores = detection[5:]
            classID = np.argmax(scores)
            confidence = scores[classID]

            # filter detections by (1) ensuring that the object detected was a person and (2) that the minimum confidence is met
            if classID == personIdx and confidence > MIN_CONF:
                box = detection[0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")
                x = int(centerX - (width / 2))
                y = int(centerY - (height / 2))

                # update our list of bounding box coordinates, centroids, and confidences
                boxes.append([x, y, int(width), int(height)])
                centroids.append((centerX, centerY))
                confidences.append(float(confidence))
```

Real-Time Detection

Detection function has 4 parameters, frame i.e. feed through camera, the YOLOv3 net, the classid (here "persons“), and the index of the output layer from the YOLO net. Input is taken through the camera and then initialized in a list for the continuous frames that will be converted to blob so we can input it to the neural net. Once the input has been given, the output from the forward layers is stored. The next step is to initialize the bounding boxes for the detected objects. A loop is iterated over the output layers and it finds out classIDs of the detected objects and the confidence of their detections.

Implementation

The first step in the algorithm is to load the appropriate paths for the coco dataset, as well as the YOLOv3 config file and weights and the neural net is created.

```
# load the COCO class labels our YOLO model was trained on
labelsPath = os.path.sep.join([config.MODEL_PATH, "coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")

# derive the paths to the YOLO weights and model configuration
weightsPath = os.path.sep.join([config.MODEL_PATH, "yolov3.weights"])
configPath = os.path.sep.join([config.MODEL_PATH, "yolov3.cfg"])

# load our YOLO object detector trained on COCO dataset
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
```

The next step is to find the output layers to be supplied as parameters to the detection function. Frames from the video streams are collected, which is another parameter to be sent to the detection function. After passing the parameters, the resulting output from the function i.e. the detection boxes are stored in a list. We iterate over the detection results and calculate distance between any possible bounding boxes and store their distances. These distances are in measures of pixels on the screen. If the measured distance violates the range of min/max safe distance, their respective centroids are appended to abnormal i.e. limiting and serious violations list. Finally, the bounding boxes are updated for the centroids that fall under limiting and serious cases to orange and red colors.

```
# ensure there are at least two people detections
if len(results) >= 2:
    # extract all centroids from the results and compute the
    # Euclidean distances between all pairs of the centroids
    centroids = np.array([r[2] for r in results])
    D = dist.cdist(centroids, centroids, metric="euclidean")

    # loop over the upper triangular of the distance matrix
    for i in range(0, D.shape[0]):
        for j in range(i + 1, D.shape[1]):
            # check to see if the distance between any two
            # centroid pairs is less than the configured number of pixels
            if D[i, j] < config.MIN_DISTANCE:
                # update our violation set with the indexes of the centroid pairs
                serious.add(i)
                serious.add(j)
            # update our abnormal set if the centroid distance is below max distance limit
            if (D[i, j] < config.MAX_DISTANCE) and not serious:
                abnormal.add(i)
                abnormal.add(j)
```

We were able to incorporate concepts like Threading to optimise the frames captured by the OpenCV algorithm to give higher FPS for real time detection and have added insightful interface messages like the safe distance to be followed as well as the monitored subjects and their violations according to the box colors.

```
class ThreadingClass:
  # initiate threading class
  def __init__(self, name):
    self.cap = cv2.VideoCapture(name)
  # define an empty queue and thread
    self.q = queue.Queue()
    t = threading.Thread(target=self._reader)
    t.daemon = True
    t.start()

  # read the frames as soon as they are available, discard any unprocessed frames;
  # this approach removes OpenCV's internal buffer and reduces the frame lag
  def _reader(self):
    while True:
      (ret, frame) = self.cap.read() # read the frames and ---
      if not ret:
        break
      if not self.q.empty():
        try:
          self.q.get_nowait()
        except queue.Empty:
          pass
      self.q.put(frame) # --- store them in a queue (instead of the buffer)
```

Chapter 4

Experimental/Simulation Results and Discussion

Below pictures show the working of the model and the metrics that we received for the trained data have all been shown below.

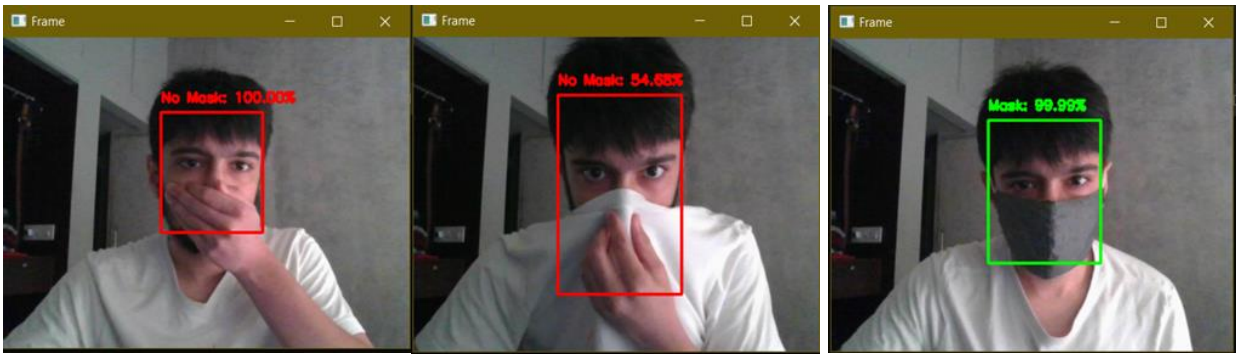

Figure 3 Real Time Detection - a,b) No Mask, c) With Mask

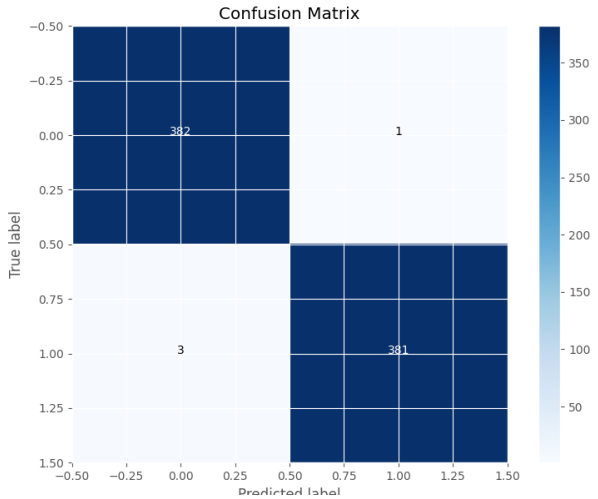

Figure 4 Confusion Matrix for trained and tested model for 20 epochs

As we can observe from the confusion matrix, we have extremely accurate results and this can be verified from the classification report depicted below.

	precision	recall	f1-score	support
with_mask	0.99	0.99	0.99	383
without_mask	0.99	0.99	0.99	384
accuracy			0.99	767
macro avg	0.99	0.99	0.99	767
weighted avg	0.99	0.99	0.99	767

Figure 5 Performance metrics for Face Mask Detection

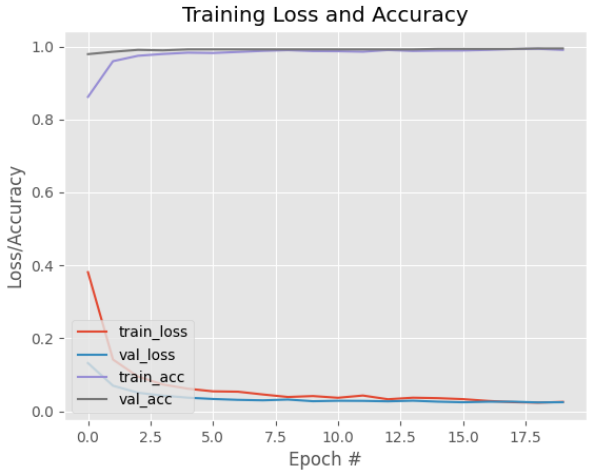

Figure 6 Training loss and Accuracy Results

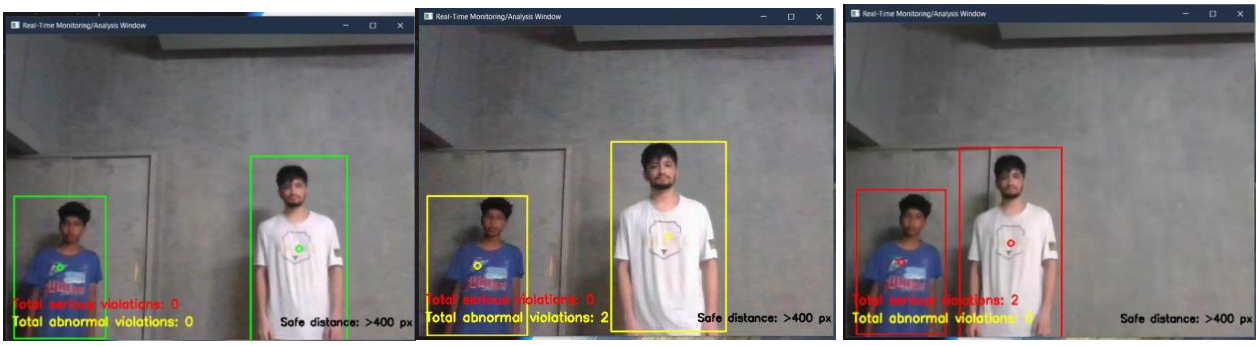

Figure 7 Real time Social Distancing Detection

In the three screenshots, we can observe three scenarios where the detected entities, (here people), are either safe distance away (that has been indicated and can be modified as per perception of camera), or about to violate the social distancing rule depicted in the yellow and red outlined boxes. The violations, be it abnormal or serious, have both been indicated in the GUI window itself and this might prove useful in larger applications containing bigger crowds of test entities.

Chapter 5

Challenges faced for model designing

Face Mask Detector Algorithm

The main objective was to remodel the existing Face Mask Detector neural net by tuning the hyperparameters to achieve better accuracy and prediction results. However, doing this was a big challenge especially with the fact that changing the learning rates as well as the layers of the neural net required a lot of hardware and time resources. Data Augmentation was also carried out to increase the size of the relatively small dataset (4K images) due to it being recent.

Social Distancing Detector Algorithm

The major challenge for this model was to program the YOLOv3 neural net to be executable in real time. So instead of the usual, processing frames stored on device, we opted to perform video capture from the device camera, simultaneously pass it to the model and make it run the neural net on it to detect social distancing between two persons. To do so, we had to go through a lot of trial and error regarding the data-scaling for the images in the video feed captured as frames, otherwise the date processed was not accurate. Secondly, we had to make it so that the frames that are processed have the correct bounding boxes and the metrics used for distance calculation which was another daunting task to do.

Another one of the challenges we faced during the implementation of the social distancing detector was the incorporation of threading to increase the FPS for the real-time capture and modelling of the images. Since that task was already very computationally heavy, incorporating threading was a minor boost to the FPS increasing it by 10 frames per second. However, the threading algorithm utilized by us is nowhere near optimized as threading in itself is a concept that requires months of research in itself.

Chapter 6

Conclusion and Future Scopes

Throughout the course of this semester, we were able to broaden our knowledge of programming acquired during the previous semesters as well as that gained from the research conducted during the initial phase of the Minor Project work, and design a working real-time system on face and social distancing detection using the resources of Python. We were able to optimize the program and integrate quality-of-life modifications like config files and threading algorithms that could helpthe user utilize this program as an application as well. The model that we designed was able to output exceptional results in terms of standard metrics like accuracy and precision of detections and predictions based and we were not only successful in implementing the concepts studied in ourresearch but also improve upon their individual performances too. The program is most suitable for solid performance on complex real-time applications. The field of object detection and even Computer Vision as a whole is ever-expanding and this project can be taken up in futurecircumstances to incorporate even more optimized algorithms as well as compatibility upgrades. Itcan also be used to research performances of different hardware boards since the algorithms used in this project are all embedded-system compatible.

References

[1] A. Yamout, A. Abdelmawgood, E. Sadick and M. Naguib, "Comparative Evaluation of Face Detection Algorithms," 2020 16th International Computer Engineering Conference (ICENCO), 2020, pp. 64-71, doi: 10.1109/ICENCO49778.2020.9357386.

[2] Sethi, Shilpa et al. “Face mask detection using deep learning: An approach to reduce risk of Coronavirus spread.” Journal of biomedical informatics vol. 120 (2021): 103848. doi:10.1016/j.jbi.2021.103848

[3] K. Bhambani, T. Jain and K. A. Sultanpure, "Real-time Face Mask and Social Distancing Violation Detection System using YOLO," 2020 IEEE Bangalore Humanitarian Technology Conference (B-HTC), 2020, pp. 1-6, doi: 10.1109/B-HTC50970.2020.9297902.

[4] W. Vijitkunsawat and P. Chantngarm, "Study of the Performance of Machine Learning Algorithms for Face Mask Detection," 2020 - 5th International Conference on Information Technology (InCIT), 2020, pp. 39-43, doi: 10.1109/InCIT50588.2020.9310963.

[5] Preeti Nagrath, Rachna Jain, Agam Madan, Rohan Arora, Piyush Kataria, Jude Hemanth, SSDMNV2: A real time DNN-based face mask detection system using single shot multibox detector and MobileNetV2, Sustainable Cities and Society

[6] Elliot Mbunge, Sakhile Simelane, Stephen G Fashoto, Boluwaji Akinnuwesi, Andile S Metfula, Application of deep learning and machine learning models to detect COVID-19 facemasks - A review, Sustainable Operations and Computers

[7] Saponara, S., Elhanashi, A. & Gagliardi, A. Implementing a real-time, AI-based, people detection and social distancing measuring system for Covid-19. J Real-Time Image Proc 18, 1937–1947 (2021).

[8] Punn, N.S.; Sonbhadra, S.K.; Agarwal, S. Monitoring COVID-19 social distancing with person detection and tracking via fine-tuned YOLO v3 and Deepsort techniques. arXiv 2020,arXiv:2005.01385.