ANKARA UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

COM 4062-H PROJECT REPORT

COVID-19 Warning System with Social Distance and Mask Detection

Ahmad Abdulkader

17290654

M. Farok Mohammed

17290639

Assoc. Prof. Dr. Hacer Yalım Keleş

May 2021

ABSTRACT

Artificial intelligence (AI) went through a huge development during the last years that today AI can be seen almost everywhere. Today, every technology can be used in a good or bad manner, in this project we show a positive side of using AI and its applications.

In 2019 a virus called COVID-19 broke out resulting in a pandemic, the virus is primarily spread between people during close contact, by wearing a mask the risk of transmission is significantly reduced.

This report starts by going over a brief introduction about the virus, current fields in AI, and how can we use AI in order to fight the virus, continuing to an introduction about the various methods that can be used to solve this specific problem and the method we chose. Finally, we go over some in-depth details about the method used and the results we achieved during the worktime of this project.

LIST OF ABBREVIATIONS

COVID-19 : Coronavirus disease 2019

ML : Machine Learning

DL : Deep Learning

ANN : Artificial Neural Networks

CNN : Convolutional Neural Networks

R-CNN : Region Based Convolutional Neural Networks

YOLO : You Only Look Once

TP : True Positive

FP : False Positive

TN : False Negative

FPS : Frames Per Second

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Overview of COVID-19

Coronavirus disease 2019 (COVID-19) is an infectious disease caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) [4]. It was first identified in December 2019. Common symptoms include fever, cough, fatigue, shortness of breath, and loss of smell and taste. While the majority of cases result in mild symptoms, some progress to more serious symptoms that can be deadly. The virus is primarily spread between people during close contact, most often via small droplets produced by coughing, sneezing, and talking. Less commonly, people may become infected by touching their face. Recommended measures to prevent infection include frequent hand washing, maintaining physical distance from others, quarantine, covering coughs, and keeping unwashed hands away from the face. The use of medical face masks has been recommended by health officials in public settings to minimize the risk of transmissions, with some authorities requiring their use. Managing the symptoms involves the treatment of symptoms, supportive care, isolation, and experimental measures. The World Health Organization declared the COVID-19 outbreak a public health emergency of international concern on 30 January 2020 and a pandemic on 11 March 2020. Local transmission of the disease has occurred in most countries. Various preventive measures are playing an important role in managing the pandemic and trying to delay additional cases until an effective treatment or vaccine become available. Preventive measures to reduce the infection include staying at home, wearing a mask in public, avoiding crowded places, keeping distance from others, washing hands with soap and water often and for at least 20 seconds, and avoiding touching the eyes, nose, or mouth with unwashed hands. Maintaining such preventive measures is difficult. Social distancing strategies provided by health officials aim to reduce contact of infected persons with large groups, but because of the long-expected vaccine, closing schools and workplaces, restricting travel, and cancelling large public gatherings for a long time is not an option. By the start of year 2021 all countries are trying to restore life by opening schools and workplaces, and adapting with the current situation. Following social distancing rules and wearing a mask are the main weapons

against COVID-19 until the vaccines under current development prove their effectiveness.

1.2. Computers to the Rescue

Today, computers are helping humans in many tasks that can save them a huge amount of time and make them focus on other tasks that might be beneficial for them. As the virus is spreading the need to wear masks and preserve social distancing is arising, but unfortunately not all precautions have been applied by all people, so the need to monitor the violations is becoming essential.

Today, surveillance systems are provided in almost every institution, but monitoring mask and social distance violations from such systems can be a tough task, especially in crowded places. With the huge advancements in AI in the last years, computers can aid in monitoring these violations, in order to accomplish this task, first we need to teach a computer how to find such violations, such a task can be solved by utilizing the power of deep learning.

1.3. Machine Learning and Deep Learning

Machine learning (ML) is a subset of artificial intelligence, which is the science of getting computers to learn and act like humans do. Machine learning uses models that are built based on mathematical algorithms and trained on sample data in order to give computers the ability to learn by themselves without being explicitly programmed to.

Deep learning, a subfield of machine learning, specializes in what is known as Artificial Neural Network (ANN) for learning from data. ANN is a class of machine learning algorithms that specialize in pattern recognition, inspired by the structure and function of the brain. The ability of ANN to learn features by itself is a huge advantage that made a path for other ANN architectures to solve a lot of problems that were hard for computers to solve in the past.

Deep learning added a huge boost to the field of computer vision, with deep learning, a lot of new applications of computer vision techniques have been introduced and are now becoming parts of our everyday lives, these include face recognition, self-driving cars, natural language processing, and object detection.

The biggest advantage of deep learning is that it added the ability to teach a computer to find and recognize objects even in real time, which makes deep learning a prefered choice when designing real time systems.

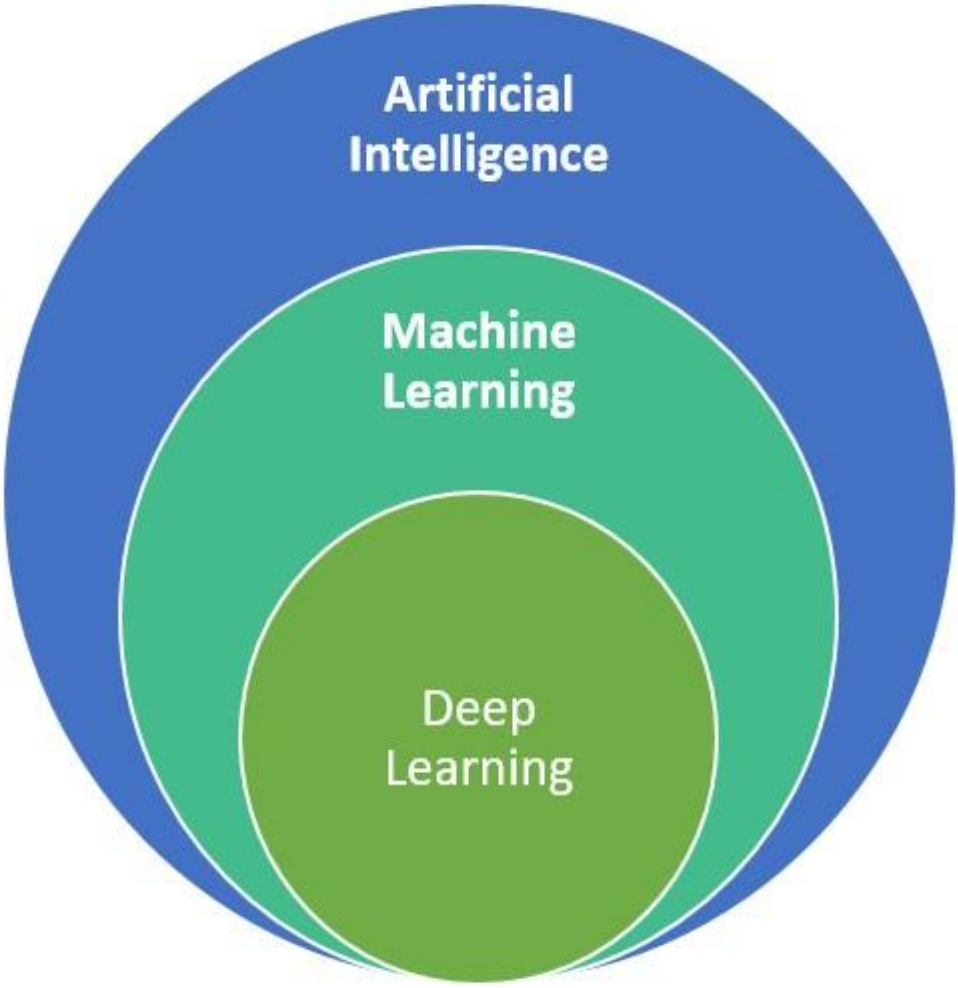

Figure 1.1. Deep learning as a subset of Machine Learning [1]

1.4. Purpose

The aim of this work is to utilize the power of deep learning to build a warning system that can perform two main tasks:

- Detect people not wearing face masks, mark them, and count the number of violations.
- Detect social distancing violations, mark them, and count the number of violations.

2. CONVOLUTIONAL NEURAL NETWORKS

A Convolutional Neural Network (CNN) is a deep learning algorithm that can take an input image and use low level features to construct a hierarchical pattern in data and assemble more complex patterns to identify objects [40], using simple artificial neural networks for computer vision problems usually is not efficient, imagine using a 64x64x1 image and feeding it to a neural network, the network will have a total of 4096 input nodes, with such number of nodes, these networks will need a huge computational power, CNNs overcome this problem by using what is called as convolutional layers.

Convolutional layers are used to reduce images into a form that is easier to process without losing features that are critical for getting a good prediction. Moreover, CNNs use what is called as pooling layers to reduce the spatial size of the convolving features, which decreases the computational power required to train the network even more. CNNs are mainly used to solve classification problems, but in the recent years more complex types of CNNs were designed, to solve object detection problems.

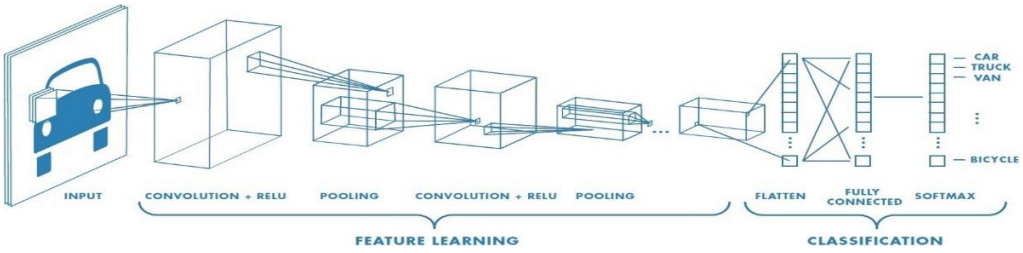

Figure 2.1. Convolutional Neural Network [34]

3. COMPUTER VISION PROBLEMS

Building a system that can identify and locate people wearing masks requires the usage of what is known as object detection. Object detection is built upon two other known problems in computer vision, classification, and localization.

3.1. Classification

To be able to find objects in images, the first goal is to teach a computer to identify what kind of object it is looking at, this problem is known as classification. Classification is a simple problem that requires the image to contain only one type of objects, usually the image will have the object cantered.

Figure 3.1 Example of image classification.

3.2. Localization

Classifying objects is an important task but it's not enough as a lot of today's application, including the purpose of this project, requires the location of an object. In order to locate an object, a convolutional neural network can be fed not only the object type, but also the middle point, width, and height of an object, using these inputs the network can learn to predict the location of the object in the image.

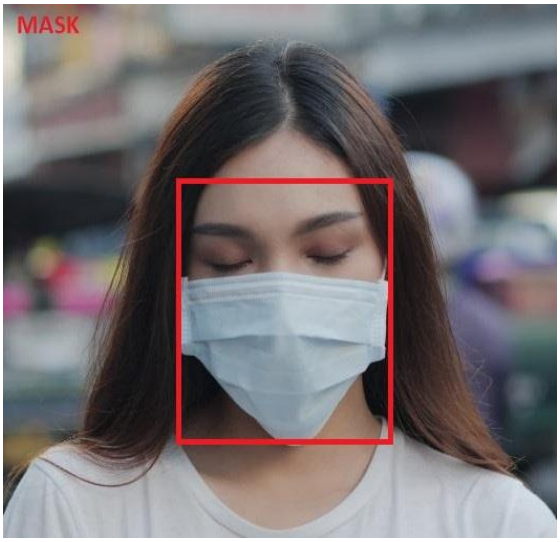

Figure 3.2. Example of object classification and localization.

3.3. Object Detection

Image classification and localization is a powerful duo, but a question arises when building a system that can find objects in images containing multiple objects, this problem is known as object detection, in the last years deep learning was used to build powerful models that can run in real-time systems and solve object detection problems.

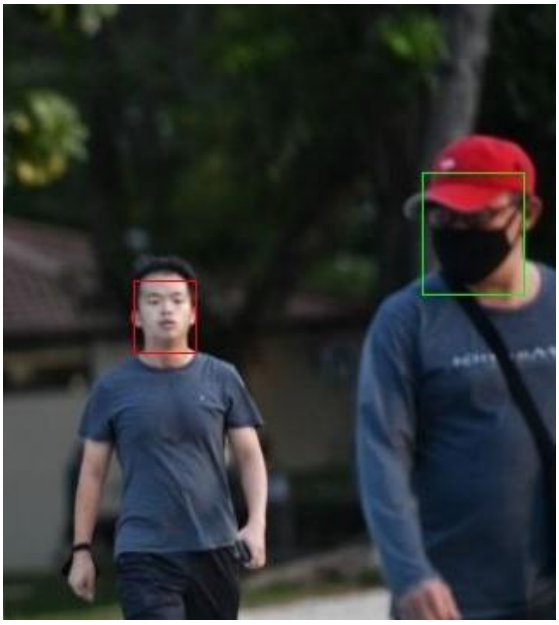

Figure 3.3. Example of object detection.

4. OBJECT DETECTION USING DEEP LEARNING

Today, three main methods for detecting objects using deep learning exist, which include sliding windows, region based neural networks, and specialized CNNs that use one forward pass to predict all the objects in an image.

4.1. Sliding Windows Detection

When deep learning first was used to solve computer vision problems, simple methods were used for solving the object detection problem, one of these methods was the sliding windows detection, this approach applies an nxn kernel to the target image and slides the kernel from left to right and top to bottom while applying a classifier to each region it passes through.

Unfortunately, this approach has many drawbacks as it takes a huge amount of time to scan only one image, applying such approach to a video is not efficient especially in real time.

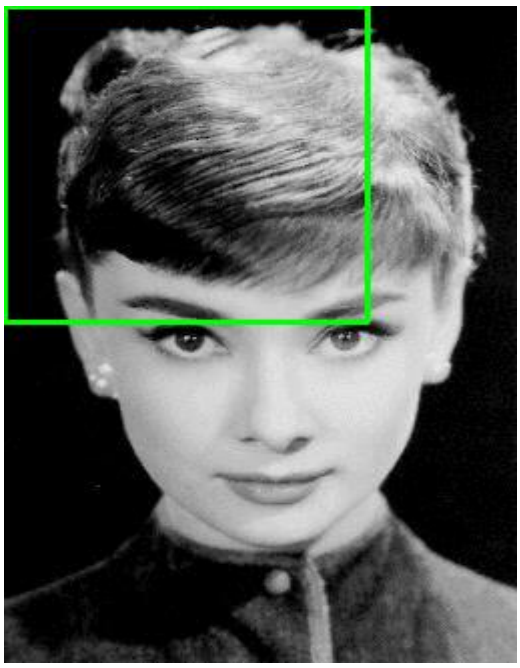

Figure 4.1. Example of sliding window detection. [3]

4.2. Region Based Convolutional Nerual Networks (R-CNN) and Variations

Rather than fully scanning an image for objects, R-CNN [30] picks number of regions from an image to run a classifier on, the network uses a segmentation algorithm on the target image, then it selects regions of interest using the selective search algorithm [36].

Figure 4.2. Example of using a segmentation algorithm on an image. [11]

R-CNN [31] is a better and faster algorithm than using sliding windows, but still not a fast algorithm to use in videos, a variation of R-CNN algorithm is fast R-CNN [31], fast R-CNN utilizes a convolutional implementation of sliding windows to classify all the picked regions giving the network a boost in speed.

Another variation is the faster R-CNN [33] algorithm, which uses convolutional neural networks to propose the regions giving the original R-CNN network even more boost in speed, but still not the speed required to run the algorithm in real-time systems.

4.3. You Only Look Once (YOLO)

YOLO [25] algorithm is a one stage detector, it differs from two stage detectors, such as R-CNN [30], by eliminating the preliminary step of detecting regions of importance and then classifying these regions. One-stage detectors are much faster models than two stage detectors, but designing such models is a complex process because such models have less accuracy than two stage detectors, thus, building such models requires a balance between speed and accuracy.

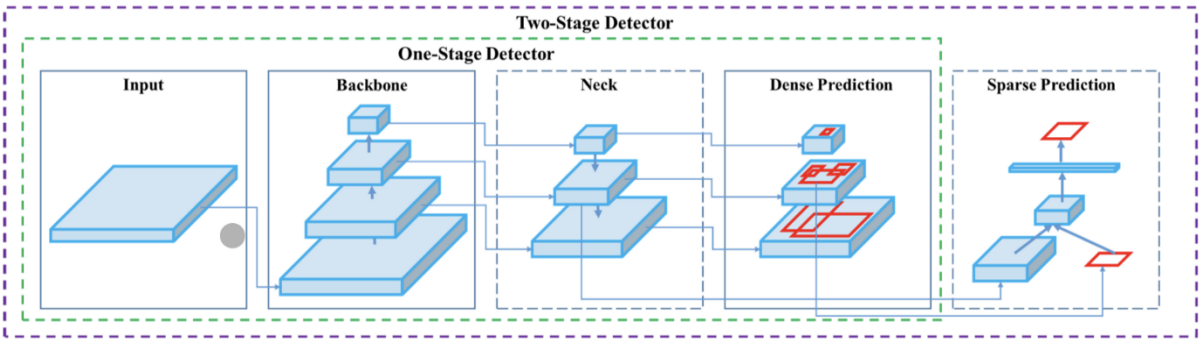

Figure 4.3. One-Stage and Two-Stage detectors. [13]

5. YOU ONLY LOOK ONCE (YOLO)

The YOLO algorithm has gone under some improvements during time, for the purposes of this project we will be using YOLO version 4 [13], still all YOLO versions are built under the same logic with some changes in its layers.

5.1. Why YOLO?

YOLO algorithm divides an image into a grid and then applies a classifier to each grid, it is designed to handle these processes in a convolutional way, simply saying, it is a big and complex convolutional neural network, the advantage of such design is that it requires only one forward pass to detect all objects in an image, thus it is called You Only Look Once. Using this unique design, YOLO is considered one of the fastest and most accurate object detection algorithms that can be used in real-time systems.

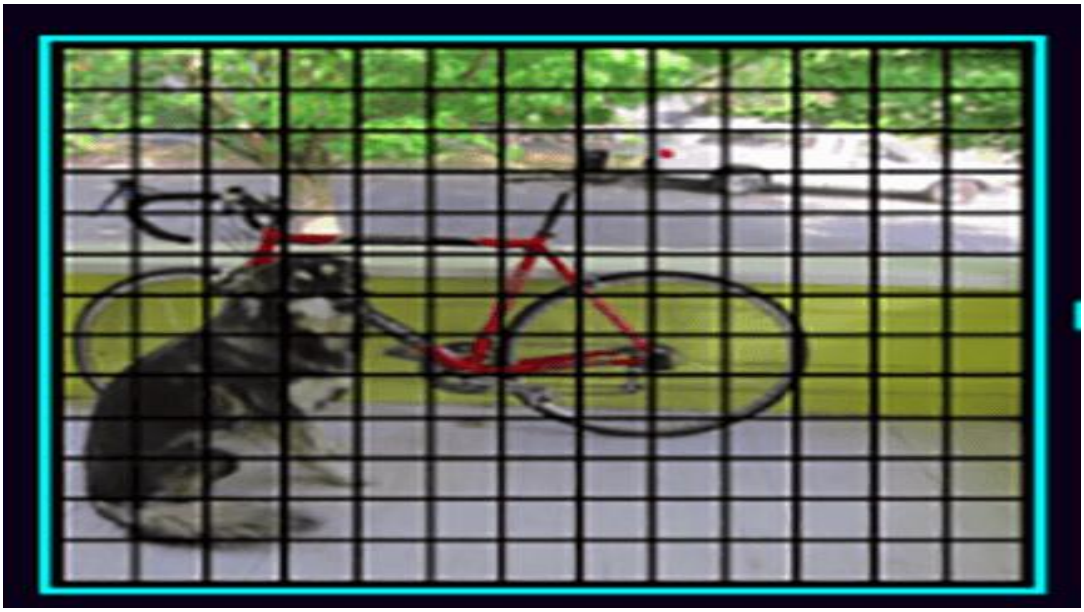

Figure 5.1. Image divided into grids. [24]

5.2. Non-max Suppression

Since YOLO uses grids to find objects, a question arises when multiple cells can find the same object.

To solve this problem, YOLO uses the Non-max Suppression algorithm. The Non-max Suppression algorithm uses the object detectors' outputs to identify overlapping objects, then it compares the accuracies of the detections and choose the detection with the highest accuracy to be the true detection.

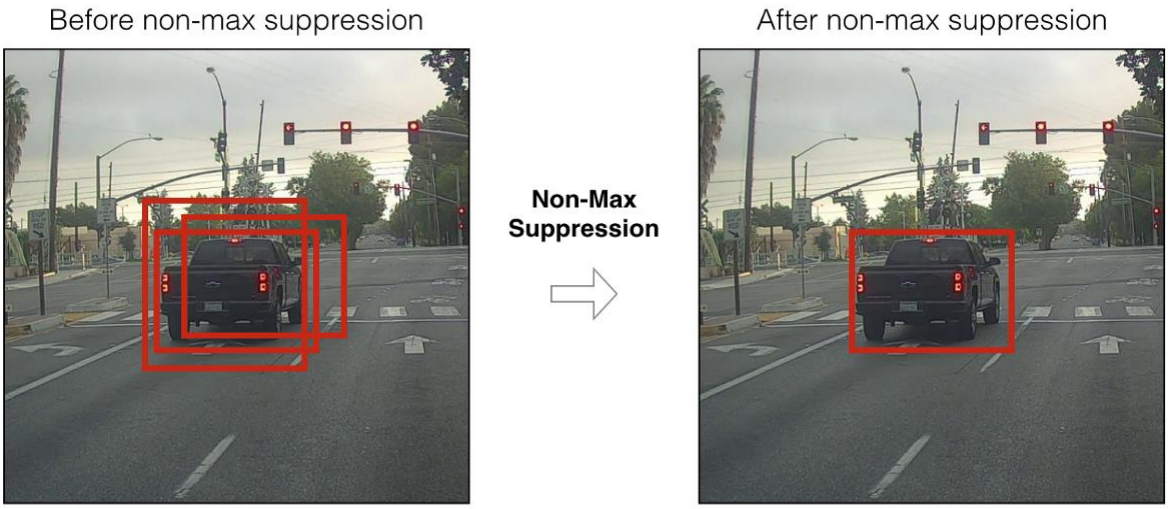

Figure 5.2. Example of applying non-max suppression. [26]

5.3. Anchors

Another reason that makes YOLO algorithm that good is using anchor boxes. Each grid in YOLO can detect only one object, to solve this problem, the network uses more than one anchor boxes with different shapes and make predictions on the same grid using different anchors, using this method each object can be assigned to a grid cell that contains the coordinates of the object and the anchor box relative to that object.

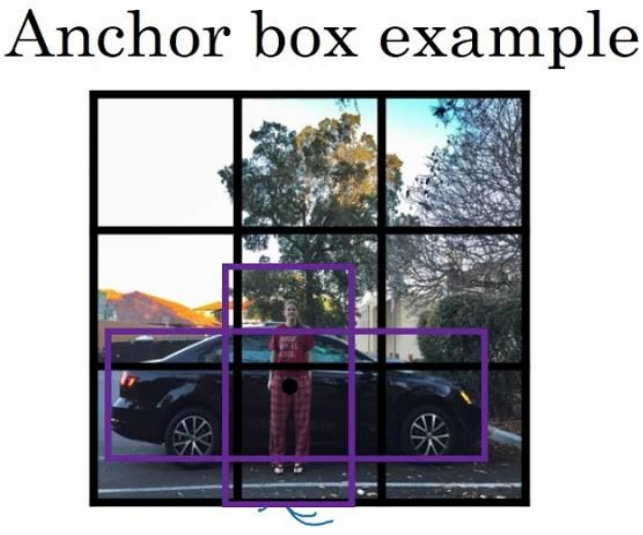

Figure 5.3. Anchor boxes. [2]

5.4. YOLOv4 Architecture

YOLOv4 [13] is big network that consists of many complicated building blocks, in this section we briefly introduce the architecture of YOLOv4 and what makes this detector special.

One-stage detectors consist mainly of four parts: Input, Backbone, Neck, and Head.

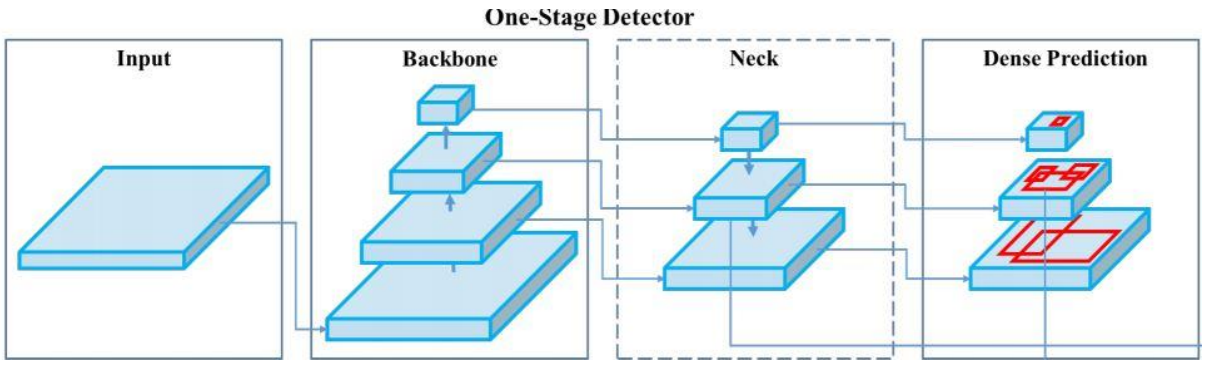

Figure 5.4. One-stage detector architecture. [13]

The authors of YOLOv4 had many choices for each part of the detector so they chose the specific parts by experimenting the performance of the detector using different parts and with the help of genetic algorithms.

The backbone of an object detector is a convolutional neural network that is able to classify objects, such backbone can be chosen from state-of-the-art classifiers available, after many experiments, the authors of YOLOv4 chose CSPDarknet53 [18] as the network's backbone. The experiments proved that CSPDarknet53 is a balance between good classification, good detection, and speed.

Recently developed object detectors insert some number of layers between the backbone and the head of the detector, such layers are used for collecting feature maps from different stages, this layer is called the neck of the detector. The neck usually contains a high input network for detecting small-sized objects, a lot of layers to cover the increased size of input network and more parameters for enhancing the model's detection ability of multiple objects of different sizes in a single image. The authors of YOLOv4 chose the Path Aggregation Network (PAN) [32] for the neck and

Spatial Pyramid Pooling (SSP) [22] as an additional module to help in achieving the previous tasks, those modules were chosen as a balance between accuracy and speed.

Finally, the head layer is where one stage and two stage detectors differ. After multiple experiments, the authors of YOLOv4 found that using YOLOv3 [23] head network is an optimal choice, and again providing balance between accuracy and speed.

The previous parts were the typical parts of any object detector, but YOLOv4 contained another set of optimization method that gave the detector a huge boost.

The first set of optimization methods is called bag of freebies, the name indicates that these enhancements are free to use as they do not affect the inference cost. The enhancements include many data augmentation methods consisting of: CutMix [35] and Mosaic data augmentation, DropBlock regularization [20], Class label smoothing [17], CIoU-loss [42], CmBN [42], Self-Adversarial Training, Eliminate grid sensitivity, using multiple anchors for a single ground truth, Cosine annealing scheduler [21], optimal hyperparameters, and Random training shapes.

The second set of optimization methods is the bag of specials, these methods affects the inference cost slightly but improve the accuracy of the object detector significantly. YOLOv4 neck is an example of the bag of specials, also the authors added other effective methods to that bag including Mish activation [19], Cross-stage partial connections (CSP) [18], and Multiinput weighted residual connections (MiWRC) [29].

Using all the previous building blocks, YOLOv4 is considered to be one of the best state-of-the-art algorithms as it achieved great test results on famous and challenging datasets.

5.5. Darknet Framework

Darknet [5] is a framework built by the authors of the YOLO algorithm using the C and C++ programming languages, the reason behind using these languages is to make the building and training of YOLO fast and efficient, also alongside C and C++, darknet framework uses parallel computing platform and application programming interface model called CUDA. CUDA was developed by NVIDIA [6] to enable users to make much faster computations using NVIDIA GPUs. Darknet models are saved using two type of files: a CFG file containing the design of the network, and a WEIGHTS file containing the weights of the neural network.

5.6. YOLOv4 Tiny

YOLO tiny [41] is a smaller version of YOLO, the algorithm contains all the optimizations methods in YOLO but differs in its backbone.

YOLO tiny uses a lighter weight backbone to minimize the number of layers and make the neural network smaller, such change in layers is a trade-off between speed and accuracy, YOLO tiny is considered to be a much faster algorithm than YOLO but has less accuracy.

Running object detectors in real time requires a big computational power, thus making YOLO hard to run on a simple GPU. YOLO tiny in contrast to YOLO can run with 30 Frames Per Second (FPS) using a simple GPU, so to make the project easier to use in various types of machines, YOLO tiny will be included in the project.

Working with YOLO tiny is no different than working with YOLO in darknet framework, in the end we will be having two different weight files for each of the two networks, loading and using them will be the same and that way working on YOLO tiny becomes a free addition to the project.

6. DATASET

Datasets in deep learning can be considered one of the most important factors in building an efficient model, and because most deep learning models require a huge amount of data to be trained, people in this field work together to create big datasets an make them available for everyone to use, therefore, solving problems that require big dataset can be made easier.

After choosing YOLO as the model to be used in this project, the next step was to find a reliable dataset to train YOLO on. The two classes of objects needed for training the model were: person not wearing a face mask and person wearing a face mask.

6.1. Collection and Labelling

Data collection is a challenging work, as you need to account for the suitable dataset size, how representative the data is for your objects and use case scenarios. The dataset we used is collected from two different datasets [14] [15] found on Kaggle [9], which is a website that provides a joint environment for people to share their own datasets.

The datasets we found include two types of classes, “no-mask” and “mask”, and each image has a corresponding annotation file that contains the class and coordinates for each object represented in the image.

Normally, the annotations are prepared in a common format that is used by most object detection algorithms, the format defines each object as its class number, top-right corner coordinates and bottom-left coordinates. YOLO on the other hand uses a different format for labelling, as it defines each object as its class number, middle point coordinates, and the width and height of the object.

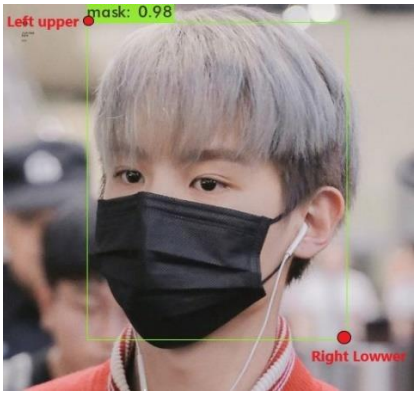

Figure 6.1. Standard labelling format.

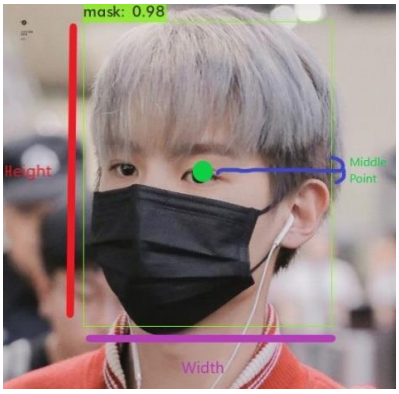

Figure 6.2. YOLO labelling format.

To overcome the problem of annotations, we built a python script to handle converting the usual format to YOLO format.

```
csv = pd.read_csv('annots.csv')

images = csv['name']
classes = csv['class']
bounding_boxes = [csv['left'], csv['top'], csv['right'], csv['bottom']]

for image, bounding_box, _class in images, bounding_boxes, classes:
    image_data = cv2.imread(image)
    img_height, img_width = image_data = cv2.imread(image).shape[:2]

    x_center = ((bounding_box[0] + bounding_box[2]) / 2) / img_width
    y_center = ((bounding_box[1] + bounding_box[3]) / 2) / img_height
    width = (bounding_box[2] - bounding_box[0]) / img_width
    height = (bounding_box[3] - bounding_box[1]) / img_height

    annot_file = os.path.splitext(image)[0] + '.txt'
    label_code = 0 if _class == 'no-mask' else 1

    if os.path.isfile(f'annots/{annot_file}'):
        with open(f'annots/{annot_file}', 'a') as file:
            file.write(
            f'\n{label_code} {x_center} {y_center} {width} {height}')

    else:
        with open(f'annots/{annot_file}', 'w+') as file:
            file.write(
                f'{label_code} {x_center} {y_center} {width} {height}')

        with open('annots.txt', 'a') as file:
            file.write(f'data/mask-dataset/images/train/{annot_file}\n')
```

Figure 6.3. Script to convert standard labelling format to YOLO format.

Another problem we faced when collecting the dataset was that nearly most of the datasets were not fully labelled, labelling such datasets may take some serious time, to overcome this problem we used a simple trick for labelling the data.

As mentioned in section 5.5, alongside using YOLO we used a small variation of it called YOLO tiny, this network can be trained with less data than YOLO needs.

Using the labelled images in the collected dataset we were able to train YOLO tiny achieving some reasonable results. Using this trained network, we built a function to make predictions on the unlabelled data and generate an annotation file for each image.

```
def create_annots(self, image_path, bounding_points, frame, label):
    img_height, img_width = frame.shape[:2]

    x_center = ((bounding_points['left'] + bounding_points['right']) / 2)
    / img_width
    y_center = ((bounding_points['top'] + bounding_points['bottom']) / 2)
    / img_height
    width = (bounding_points['right'] - bounding_points['left']) /
    img_width
    height = (bounding_points['bottom'] - bounding_points['top']) /
    img_height
    label_code = 0 if label == 'no-mask' else 1

    file = image_path.split('/')[-1]
    file_name, ext = os.path.splitext(file)


    if os.path.isfile(f'{file_name}.txt'):
        with open(f'{file_name}.txt', 'a') as file:
            file.write(
            f'\n{label_code} {x_center} {y_center} {width} {height}')


    else:
        with open(f'{file_name}.txt', 'w+') as file:
            file.write(
                f'{label_code} {x_center} {y_center} {width} {height}')

        with open('train.txt', 'a') as file:
            file.write(f'data/mask-dataset/images/test/{file_name}{ext}\n')
```

Figure 6.4. A method for generating annotation fiels using YOLO tiny.

Using the previous method, we were able to generate enough data to train YOLO on. This method is not perfect and can produce some problems, in the next section we introduce the problems we faced regarding the dataset and how can we improve it in the future for even better results.

6.2. Problems and Improvements

More data always means better results in deep learning, the dataset used to train YOLOv4 for this project currently contains 6000 thousand images in total, divided into train and test sets. In the future we plan on collecting more images as it will improve our detector's performance significantly.

Building datasets that contain faces of people is considered a hard task because it's hard to find such public images that their owners approve on using these images for such applications, therefore, finding such datasets on Kaggle was a boost for the project.

One problem with the datasets available is the perspective of the images, as shown in figure 6-2, most of the images, similar to this image, are taken from the ground and with normal cameras, such images contain two problems regarding the application of the project, the first problem is that a lot of the faces in the background are blurred, such faces cannot be used in training because they might corrupt the features learned by the detector, also as we will discuss in the results section, these images can affect the test results of the detector. This project is built to be used in systems like surveillance cameras, therefore, training YOLO on images taken by such cameras can make a huge difference in the detector's accuracy.

Figure 6.5. Example of blured objects.

Another problem we discovered when exploring the dataset is the amount of white and blue masks in the dataset, such inequality between these colours and other colours can make the detector biased to white and blue masks, to solve this problem we consider adding more colourful masks to the dataset.

As we start adding mask images in the future, we need to preserve the balance of the dataset, so whenever an image containing people wearing face mask is added, a corresponding image containing people not wearing face masks will be also added from the famous WIDER [16] dataset.

Finally, because the dataset was labelled using YOLO tiny, testing YOLO tiny on this dataset will be unreliable, also the network resulted in some wrong labels that affected the training process of YOLO, to solve this problem we consider labelling the dataset manually, this process will take a lot of time so we consider it as a big future improvement that might enhance the performance of both YOLO and YOLO tiny.

6.3. Final Dataset

After long period of work, the current dataset was fully relabelled, in addition, during the labelling process, multiple new images were added to the current dataset to solve some problems, like the lack of coloured masks in the dataset. The final dataset's split used for training the mask detector is shown in the following graph:

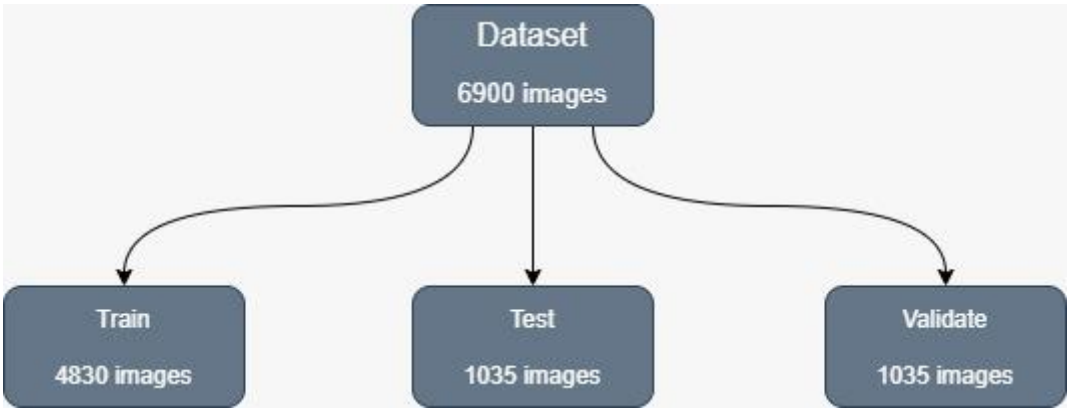

Figure 6.6 Final dataset split.

7. MODEL BUILDING

7.1. Environment

Training YOLO using the darknet framework is a simple task as darknet provides a simple interface for training and evaluating YOLO, but setting up darknet is relatively a hard task and takes a lot of time, to make the project more efficient and easier to setup on all devices, we decided to train YOLO using the darknet framework, and then use the weights we obtain to write and interface using python programming language.

Python is one of the most used language for deep learning projects, as it is easy to use and includes many frameworks and libraries that can boost the training and usage of deep learning models.

Tensorflow [7] among many frameworks is one of most preferred frameworks to use in deep learning projects, tensorflow also provides a high-level interface called keras [10], keras provides a simple interface that enable users to use the computational power of tensorflow using simple and abstract pieces of code.

7.2. Darknet Weights to Tensorflow Weights

Tensorflow framework provides a specific format for saving a neural networks architecture and weights, the problem is darknet framework uses a very different type of files to save YOLO's architecture and weights, to overcome this problem, a lot of open source implementations of YOLO are provided, using one of these implementations [8] we were able to convert the results of training YOLO using darknet framework to a tensorflow format, thus making YOLO available to use in python and making the project more scalable and easier to setup and use.

```python
import time
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model


class YoloMask:
    def __init__(self, detector_path):
        loaded_detector = load_model(detector_path, compile=False)
        self.mask_detector = loaded_detector.signatures['serving_default']

    def detect_from_image(self, image_path):
        frame = cv2.imread(image_path)

        if frame is None:
            print('\n[WARNING]: Please enter a valid image path')
            return

        self.__detect_frame(frame)

        if self.__write_detection:
            cv2.imwrite('prediction.jpg', frame)
```

Figure 7.1. YoloMask class.

In the figure 7-1 is a snapshot from the source code, the image illustrates the ease in using YOLO in python as it requires only two lines of code to load the network architecture and weights using tensorflow framework.

8. TRAINING AND CLOUD COMPUTING

Training neural networks is a task associated with a powerful hardware demand, specifically training neural networks requires a strong GPU. Rather than setting up a GPU, an alternative is cloud computing.

8.1. What is Cloud Computing

In the recent years, a new technology known as cloud computing has emerged. Cloud computing is the set of multiple services, like servers, databases, and analytics, delivered over the internet. Rather than customizing a setup for each type of project, a developer can use cloud computing services and save a lot of time, and in most cases, lower the expenses of the project. Today, three major cloud providers exist, Google,

Microsoft, and Amazon. In this project, we use google's provided services, specifically we incorporate the power of the google colab platforms. Google colab service provides machines for customers, these machines have the option of including a GPU in it, google provides some of its most powerful GPUs in this platform.

8.2. Training

Our current hardware enabled us to train YOLOv4 for 10,000 iteration in an estimated time of 70 hours. Using cloud computing, we were able to train YOLOv4 for 10,000 iterations in only 15 hours. With this huge difference in training hours, we were able to train YOLOv4 and YOLOv4 tiny multiple times using different configurations and different datasets to get the best results.

9. SOCIAL DISTANCING DETECTION

Social distancing is a measure used in closed public places to prevent the formation of crowds. In a social distancing detection system, the first objective is to detect pedestrians, then a process of mapping pixels to real world measurements must be applied, using the previous two processes, distances between objects can be calculated. Finally, additional features like bird-eye view and camera calibration can be applied to enhance the distance measurement even more.

9.1. Pedestrian Detection

Pedestrian detection is one of the most used applications in object detection as pedestrians are considered a common object to detect and apply multiple and different applications on. Rather than spending time training an object detector to detect pedestrians, multiple trained detectors are available and ready to use. YOLO, which is the detector we are using in the project, is already trained on the COCO dataset [37], this dataset contains more than 80 different objects including pedestrians, using the trained weights, we can incorporate YOLO to detect pedestrians and solve the first part of the social distancing functionality.

9.2. From Pixels to Real-World Measurements

After performing pedestrian detection, the next step is to calculate the distance between each pedestrian to infer whether a person is abiding the social distancing rules or not. Calculating distances between any two objects can be simply done by calculating the Euclidean distance between some points that are in the pedestrians' bounding boxes.

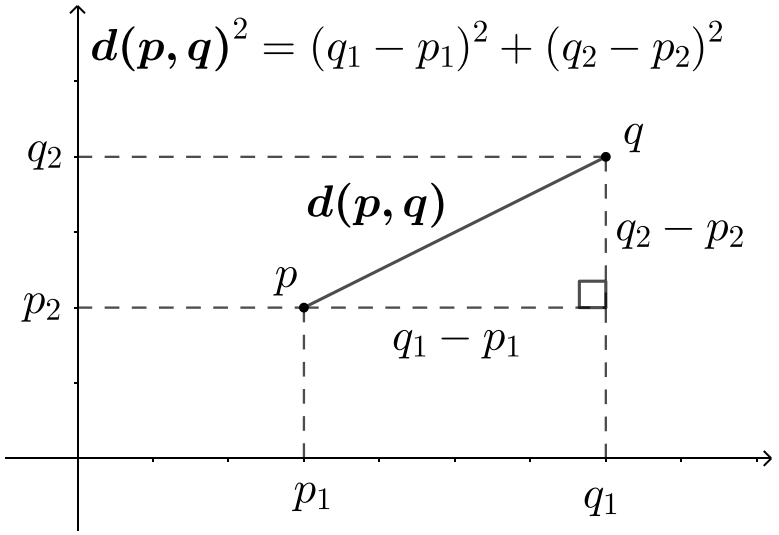

Figure 9.1. Euclidean distance. [39]

Images are represented as pixels, thus, the points that are used to calculate the distance between any two objects is represented in pixels, using such method can lead to many inaccurate results. To enhance the accuracy of the distances, the camera using the system should be stationary, then, an image containing a pedestrian using the same camera must be provided to the system, alongside the pedestrian height in real-world measurements, such as meters. Using the previous information, the system can use a simple proportion to map pixels to meters, thus, the system can calculate the distances more accurately.

9.3. Bird's-eye View

Mapping pixels to a real-world metric using the previous method is accurate enough, however, in a normal camera perspective the geometric properties of objects change

according to the distance from the point-of-view; this may result in an unreliable messurment.

More accurate messurments can be achieved by applying a Bird's-eye View transformation to the scene. Bird's-eye View transformation provides an estimated top-down view of the scene, and thus, provides similar geometric properties for all objects, regardless of their distance and location.

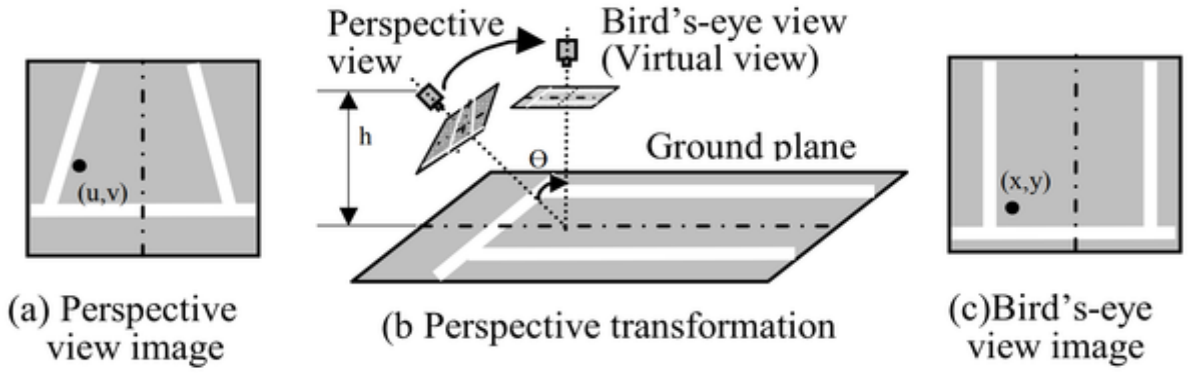

Figure 9.2. Bird's-eye view [29]

Implementing such a transformation requires four corner points from the scene we need to detect social distancing in, these points specify the scene's surface that the transformation will be applied on. Moreover, it should be noted that this method assumes that the camera is fixed and the four points chosen to generate a perspective containe a plane where all objects in it are on the same surface. Applying this can lead to a more robust results.

10. SYSTEM ARCHITECTURE

Due to the nature of the project, we split the system into two micro-systems: social distancing and mask detection. The reason behind the split is that each functionality requires different type of setup, the mask detection system is a plug and run system, whilst the social distancing detection system requires preliminary steps before running. Moreover, the social distancing system must be run from a fixed camera because we are calculating some perspective variables related to the camera view, on the other hand, the mask detection system can work in a not fixed camera.

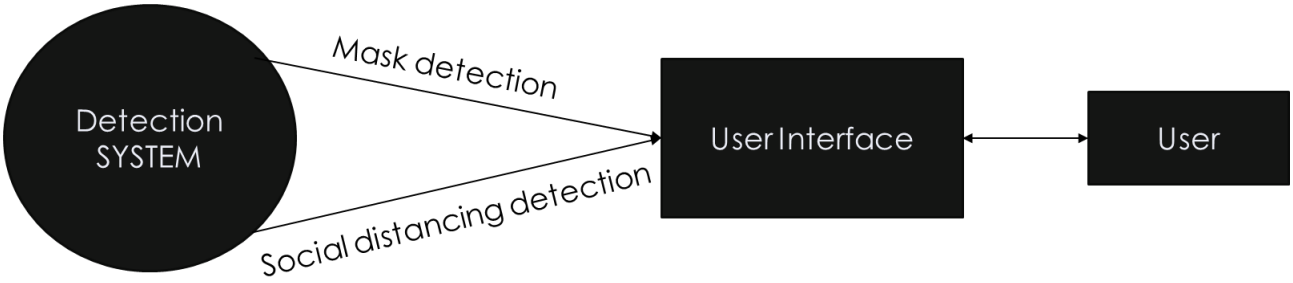

Figure 10.1. System architecture.

11. OBJECT DETECTION METRICS

Before going over the results of the project, this section will briefly introduce the metrics that were used to evaluate the results. Three main definitions affect the various metrics used which are:

True Positive (TP) -> A valid detection

False Positive (FP) -> An invalid detection

False Negative (FN) -> Ground truth not detected by the model

11.1. Intersection Over Union (IOU)

The IOU is one of the main and unique metrics used in object detection, as object detection tries to find the bounding boxes of an object, IOU evaluates how good the model is in drawing those bounding boxes.

As its name says, IOU is calculated using a simple formula:

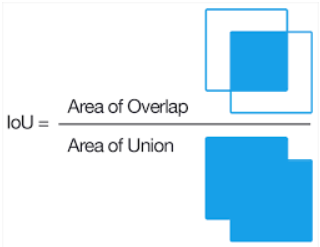

Figure 11.1. IOU formula. [3]

To evaluate the efficiency of a model, the average IOU for all detections is calculated in the end, usually an IOU equal or greater than 0.5 is considered as good detection, this value is a convention that most practitioners in object detection use, still for more accurate detection a bigger threshold can be used.

Figure 11.2 IOU. [3]

11.2. Precision and Recall

Precision and recall are one of the most used metrics in various types of computer vision problems, these metrics are simple to calculate and powerful in evaluating the performance of a model.

Precision can be defined as the ability of a model to predict the true class of an object and it can be calculated by dividing true detections (true positives) by all detections (true positives + false positives).

Recall is the ability of a model to find all the ground truth bounding boxes and it can be calculated by dividing true detections (true positives) by all ground truths (true positives + false negatives).

For both models having values closer to 1 means having less false positives and less false negatives, which concludes a better performance by the model.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Figure 11.3. summary of precision and recall formulas

11.3. Precision Recall Curve (PR Curve), and Average Precision

The PR curve is combination of recall and precision, it's a plot of precision as a function of recall. This metrics can show the trade-off between precision and recall meaning it illustrates the relationship between true positives, false positives, and false negatives.

After calculating the PR curve for each class, another metric is calculated by averaging the precision across all recall values between 0 and 1, this metric is called the average precision (AP). AP helps in calculating the final and most important metric that is usually used to compare different detectors' performances.

11.4. Mean Average Precision (mAP)

As its name says, mAP is the mean of the average precision for each class defined in the detector, calculating the mAP involves calculating the precision, recall, PR curve, and average precision, meaning that the mAP can be considered as a final metric for evaluating the detector's performance and the closer the value of mAP to 1 the better the detector is.

Usually, the IOU threshold used in detection is specified when showing the mAP value in the following format:

- mAP@α, where α is the value of the IOU threshold.

12. RESULTS and DISCUSSION

In this section, we go over the results obtained by both the mask and the social distancing detector. First, we show the metrics mentioned in the metrics section and the results obtained by training yolov4 for mask detection, moreover, images tested using the mask detector are shown. Finally, we go over the results of the social distancing detection functionality, the metrics for the detector are overlooked because the detector was already trained, therefore, we go over some images obtained by testing the social distancing detector, and some of the problems of both the mask detection and social distancing detection functionalities.

12.1. Mask Detection

After training YOLOv4 and YOLOv4 tiny multiple types using different techniques, we were able to achieve great results. The detector was able to achieve a precision of 0.95, that is considered a high precision value. Moreover, the detector achieved a recall of 0.97, meaning the detector was able to find most of the ground truths in the test dataset. Finally, the detector achieved an average IOU 81.76% at 50% IOU threshold. Below is a table comparing the metrics achieved by training YOLOv4 on the old dataset, YOLOv4 on the final dataset, and YOLOv4 tiny on the final dataset.

Tabel 12.1. Testing results of YOLOv4.

Model/Metrics	Precision	Recall	IOU	mAP@50
yolov4 (old dataset)	90%	88%	74%	88%
yolov4 (new dataset)	95%	97%	81.76%	98.29%
yolov4 tiny (new dataset)	93%	96%	77.42%	97.44%

As shown in the table, both YOLOv4 and YOLOv4 tiny achieved great results after training on the final dataset. The next images are some of the results of testing the mask detection system.

Figure 12.1. Mask detection results part 1.

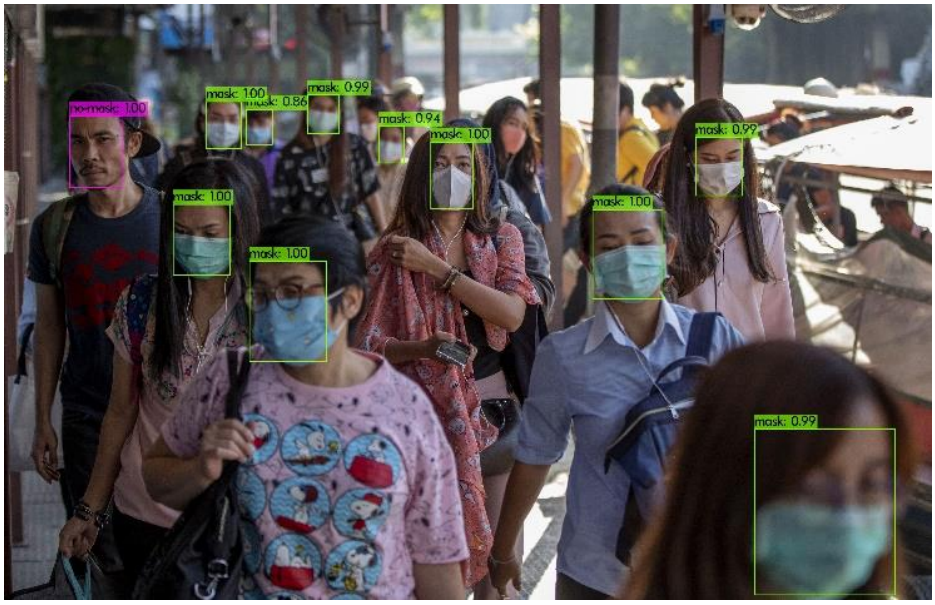

Figure 12.2. Mask detection results part 2.

Figure 12.3. Mask detection results part 3.

As shown in the images, the system is working well for most cases, but the system still has some problems to overcome. People using their hands to cover their mouths are most of the time confusing the detector, to fix such an issue, the detector must be trained on images containing such scenarios. Moreover, the detector can sometimes struggle with side faces as it was mainly trained to detect frontal face, such an issue can be solved by adding more training images and labeling peoples' side faces. Finally, more data always mean better accuracy, building bigger dataset that contains multiple scenarios from the setting we want to detect people in can boost the efficiency of the detection and reduce its errors.

12.2. Social Distancing Detection

Using the methods that were mentioned in the social distancing section we were able add the fumcntionality of detecting people who are not applying the social distancing rules, below are some images taken from a video that was processed using the social distancing detection system.

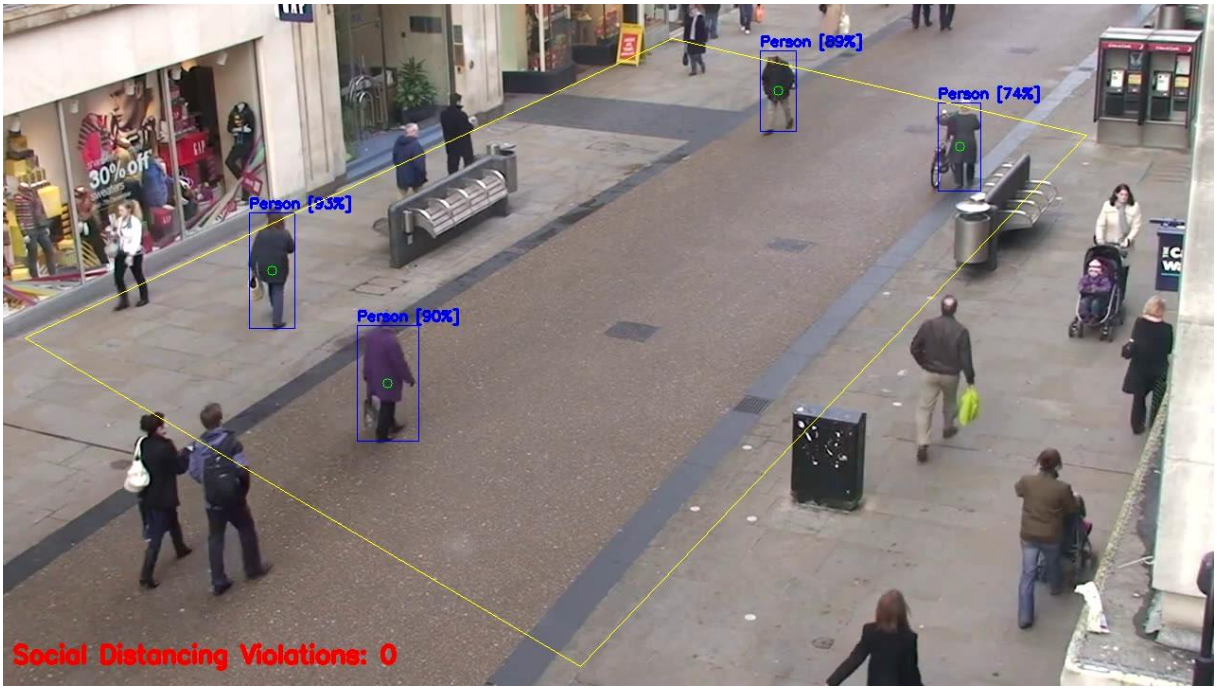

Figure 12.4. Social distancing detection results part 1.

Figure 12.5. Social distancing detection results part 2.

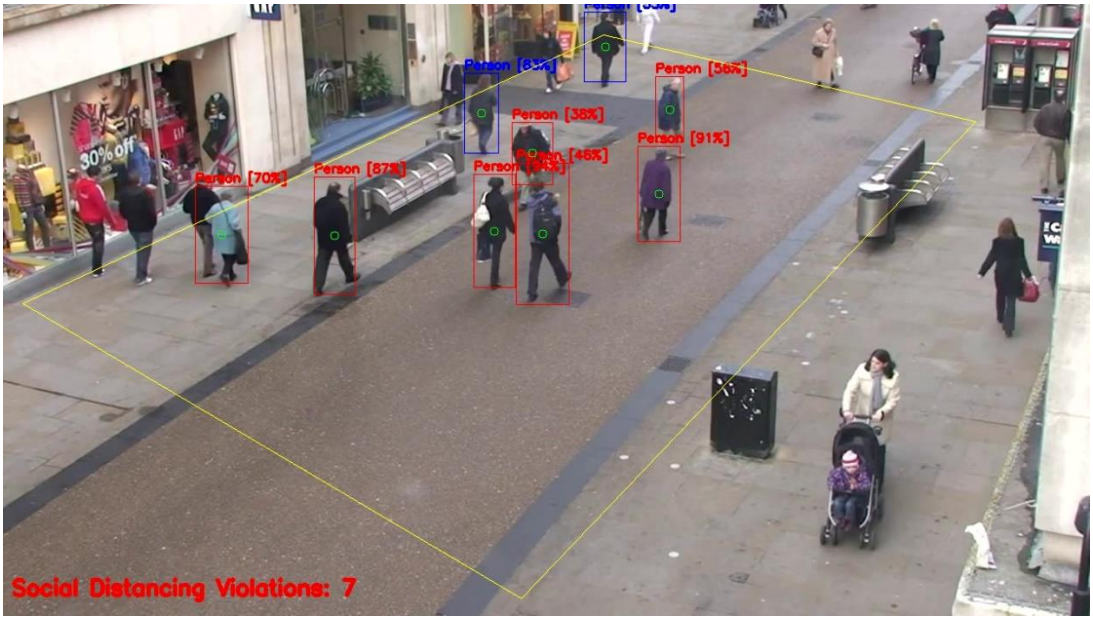

Figure 12.6. Social distancing detection results part 3.

12.3. FPS Evaluation

After stating the results and the future improvements than can be applied to boost the system's efficiency, we provide a table containing how many frames can the system process per second, the results were aquired by testing the system on multiple GPUs.

Table 12.2. FPS (Frames Per Second) evaluation.

System	Nvidia GTX1050	Nvidia Tesla P4	Nvidia Tesla T4	Nvidia Tesla P100
yolov4 Mask detection	8 FPS	16 FPS	25 FPS	36 FPS
yolov4 tiny Mask detection	35 FPS	76 FPS	70 FPS	NA
yolov4 Social distancing detection	8 FPS	15 FPS	21 FPS	34 FPS
yolov4 tiny Social distancing detection	33 FPS	56 FPS	54 FPS	NA

12.4. Discussion

Both systems achieved real-time detection using specific hardware. In addition, YOLOv4 tiny is provided to run the system on low specifications hardware. To improve the mask detection system in future works, more data can be added to the dataset, the current model still have some problems for detecting the side profile of the face. In addition, when objects like mobile phones, or a hand is used to cover the mouth and the nose, the system can be tricked to think that a person is wearing a face mask. As for the social distancing violation system, a method called camera calibration can be used to acquire the properties of the camera, the current results are acceptable, but using camera calibration we can calculate the exact distance between pedestrians.

13. CODE and USER INTERFACE

Preparing a dataset and training the model are the essential parts of any machine learning project, but when those tasks are done, a new task is required, which is providing a suitable interface for using the model built. Different people can use such a model in different ways, therefore, building a dynamic interface is also an essential part.

During the project, we were able to develop a simple interface [12] for using the system, the interface allows the users to use both the mask and social distancing detection functionalties. Furthermore, the system can be customly configured using a configuration file shipped with the interface. Finally, alongside comments, we provide a documentation file for making the system easier to use.

14. CONCLUSION

Using deep learning, we were able to build a system that can detect people wearing masks and people not wearing masks, the project started by exploring the various ways of detecting objects using computers and deep learning, after choosing a model the main objective was to collect a reliable dataset to train the model on. After searching for such datasets, we were able to build a small dataset to train YOLO tiny, a small model that we used in our project alongside the main model that was YOLO.

After training YOLO tiny the model was used to label the remaining images that were missing labels in the dataset we built, later, the complete dataset was used to train YOLO. By testing YOLO on the test data, the results achieved were good, yet the current dataset still has some problems and by improving it, the performance of the detector can be improved even more.

To improve the current results, we collected more data and relabelled the images from scratch, therefore, the results of the mask detection system improved significantly, and we were able to increase the mAp@50 from 88% to 98.29%.

In addition to mask detection, we implement pedestrian detection using YOLO detector and use scaling methods to map pixels to meters, using this data we calculate the Euclidian distance to the detected objects to find social distancing violations. To improve the results, Bird’s-eye view method is also used to make sure that all detected object has the same geometric properties.

Finally, we were able to achieve real-time detection in both mask detection and social distancing violation detection using YOLOv4 with specific hardware requirements. Whereas YOLOv4 tiny achieved real-time detection using simple hardware requirements.

BIBLIOGRAPHY

[1] "Basic Concepts of Artificial Intelligence, Machine Learning, Deep Learning." Novelty Bilişim, 2020. [Online]. Available: https://noveltybilisim.com.tr/machine-learning/basic-concepts-of-artificial-intelligence-machine-learning-deep-learning/

[2] "C4W3L08 Anchor Boxes", YoutubE, 2017. [Online]. Available: https://www.youtube.com/watch?v=RTlwl2bv0Tg

[3] "Computer Vision, Deep Learning, and OpenCV.", PyImageSearch. [Online]. Available: https://www.pyimagesearch.com/

[4] "Coronavirus Disease (COVID-19)", cdc.gov [Online]. Available: https://www.cdc.gov/media/dpk/diseases-and-conditions/coronavirus/coronavirus-2020.html. [Accessed: Jan- 2021]

[5] "darknet", GitHub, 2019. [Online]. Available: https://github.com/AlexeyAB/darknet. [Accessed: Jan- 2021]

[6] "NVIDIA: World Leader in Artificial Intelligence Computing", NVIDIA. [Online]. Available: https://www.nvidia.com/en-us/

[7] "TensorFlow". [Online]. Available: https://www.tensorflow.org/

[8] "tensorflow-yolov4-tflite", GitHub, 2021. [Online]. Available: https://github.com/hunglc007/tensorflow-yolov4-tflite. [Accessed: Jan- 2021]

[9] "Kaggle: Your Machine Learning and Data Science Community", Kaggle. [Online]. Available: https://www.kaggle.com/.

[10] "Keras: the Python deep learning API", Keras.io. [Online]. Available: https://keras.io/

[11] "What is the Difference Between Image Segmentation and Classification in Image Processing?", 2019. [Online]. Available: https://medium.com/cogitotech/what-is-the-difference-between-image-segmentation-and-classification-in-image-processing-303d1f660626

[12] A. Abdulkader and M. Farok, "Covid-19-Warning-System", GitHub. [Online]. Available: https://github.com/parot-99/Covid-19-Warning-System

[13] A. Bochkovskiy, C. Wang and H. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection", arXiv.org, 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[14] A. Lorenzo, "MaskDetection at YOLO format", Kaggle.com, 2020. [Online]. Available: https://www.kaggle.com/alexandralorenzo/maskdetection

[15] A. Purohit, "Face Mask Dataset (YOLO Format)", Kaggle.com, 2020. [Online]. Available: https://www.kaggle.com/aditya276/face-mask-dataset-yolo-format

[16] C. Change, Xiaoou, Tang, Loy, Ping, Luo, Shuo and Yang, "WIDER FACE: A Face Detection Benchmark", 2016. [Online]. Available: http://shuoyang1213.me/WIDERFACE/

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision", arXiv.org, 2015. [Online]. Available: https://arxiv.org/abs/1512.00567

[18] C. Wang, H. Liao, I. Yeh, Y. Wu, P. Chen and J. Hsieh, "CSPNet: A New Backbone that can Enhance Learning Capability of CNN", arXiv.org, 2019. [Online]. Available: https://arxiv.org/abs/1911.11929

[19] D. Misra, "Mish: A Self Regularized Non-Monotonic Activation Function", arXiv.org, 2019. [Online]. Available: https://arxiv.org/abs/1908.08681

[20] G. Ghiasi, T. Lin and Q. Le, "DropBlock: A regularization method for convolutional networks", arXiv.org, 2018. [Online]. Available: https://arxiv.org/abs/1810.12890

[21] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts", arXiv.org, 2016. [Online]. Available: https://arxiv.org/abs/1608.03983

[22] J. Sun, K. He, X. Zhang and S. Ren, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition", arxiv.org, 2014. [Online]. Available: https://arxiv.org/abs/1406.4729

[23] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement", arXiv.org, 2018. [Online]. Available: https://arxiv.org/abs/1804.02767v1

[24] J. Redmon and A. Farhadi, "YOLO: Real-Time Object Detection", Pjreddie.com, 2018. [Online]. Available: https://pjreddie.com/darknet/yolo/

[25] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", arXiv.org, 2015. [Online]. Available: https://arxiv.org/abs/1506.02640

[26] K. Sambasivarao, "Non-maximum Suppression (NMS)", Medium, 2019. [Online]. Available: https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c

[27] M. Abdi and S. Nahavandi, "Multi-Residual Networks: Improving the Speed and Accuracy of Residual Networks", arXiv.org, 2016. [Online]. Available: https://arxiv.org/abs/1609.05672

[28] R. Adrian, 2014. 4 Point OpenCV getPerspective Transform Example - PyImageSearch. [online] PyImageSearch. Available at: https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/

[29] R. Basile, 2020. A social distancing detector using a Tensorflow object detection model, Python and OpenCV. [online] Deepnote. Available at: https://deepnote.com/@deepnote/A-social-distancing-detector-using-a-Tensorflow-object-detection-model-Python-and-OpenCV-KBcEvWejRjGyjy2YnxiP5Q

[30] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", arXiv.org, 2013. [Online]. Available: https://arxiv.org/abs/1311.2524

[31] R. Girshick, "Fast R-CNN", arXiv.org, 2015. [Online]. Available: https://arxiv.org/abs/1504.08083

[32] S. Liu, L. Qi, H. Qin, J. Shi and J. Jia, "Path Aggregation Network for Instance Segmentation", arXiv.org, 2018. [Online]. Available: https://arxiv.org/abs/1803.01534

[33] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", arXiv.org, 2016. [Online]. Available: https://arxiv.org/abs/1506.01497

[34] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way", 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[35] S. Yun, D. Han, S. Oh, S. Chun, J. Choe and Y. Yoo, "CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features", arXiv.org, 2019. [Online]. Available: https://arxiv.org/abs/1905.04899

[36] T. Gevers and J. Uijlings, "Selective Search for Object Recognition", 2013. [Online]. Available: https://www.researchgate.net/publication/262270555_Selective_Search_for_Object_Recognition

[37] T. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. Zitnick and P. Dollár, "Microsoft COCO: Common Objects in Context", arXiv.org, 2021. [Online]. Available: https://arxiv.org/abs/1405.0312

[38] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu and A. Berg, "SSD: Single Shot MultiBox Detector", arxiv.org, 2015. [Online]. Available: https://arxiv.org/abs/1512.02325

[39]Wikipedia. n.d. Euclidean distance. [online] Available at: https://en.wikipedia.org/wiki/Euclidean_distance

[40] Y. Lecun and Y. Bengio, "Convolutional Networks for Images, Speech, and Time-Series", 1998. [Online]. Available: https://www.researchgate.net/publication/2453996_Convolutional_Networks_for_Images_Speech_and_Time-Series

[41] Z. Jiang, L. Zhao, S. Li and Y. Jia, "Real-time object detection method based on improved YOLOv4-tiny", arXiv.org, 2020. [Online]. Available: https://arxiv.org/abs/2011.04244

[42] Z. Yao, Y. Cao, S. Zheng, G. Huang and S. Lin, "Cross-Iteration Batch Normalization", arXiv.org, 2020. [Online]. Available: https://arxiv.org/abs/2002.05712

[43] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye and D. Ren, "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression", arXiv.org, 2019. [Online]. Available: https://arxiv.org/abs/1911.08287