Social Distancing Detection: Project report

Tequila: Tzu-Ching Yen, Muneeb Ansari, Gang Zhao

1 Abstract

Social distancing has been shown to be one of the most effective measures to slow the spread of COVID-19[1]. However, there are a number of challenges in monitoring social distancing via cameras. One needs to first identify the pedestrians and their locations from videos or images, and further account for effects of the 2D projective transformations of cameras to accurately compute interpersonal distances [2–4]. In our project, we are using the pre-trained YOLOv3 model to detect the pedestrians and then perform a homography transformation on their pixel coordinates to measure interpersonal distances [5]. Once we computed the distances and identified the pedestrians who violate social distancing measure, we plotted red bounding boxes around them.

2 Introduction and Literature Review

2.1 Introduction

While social distancing can be effective against the spread of COVID-19, it is often difficult to enforce the measure in practice. Either due to the difficulties for people in estimating interpersonal distances or the lack of awareness of the general public, social distancing violations are widespread and difficult to monitor by police forces.

On the other hand, surveillance system that store video footage of public spaces may raise concerns for the privacy of pedestrians [3]. Thus, it is favorable to develop software that is capable of real-time detection of social

distancing violations. Identifying violations using this software, a system can then sound an alarm to warn the area or report persistent violations to the law enforcement [2].

In this project, we built such software. We used the pre-trained model of YOLOv3 to detect locations, pixel coordinates of the pedestrians in videos or images. Then, we compute the interpersonal distances by applying the inverse homography transformations that are built from assumed knowledge of the internal and external parameters of the cameras [6]. Our code is publicly available on github.

2.2 Pedestrian detection model base on YOLOv5

We extracted and merged the pedestrian data from the VOC dataset and the Coco dataset. The motivation for doing so was to increase the robustness of the model from the perspective of expanding the dataset. We also performed edge adaptive scaling on the obtained dataset. Because the limit of YOLOv5's detection target is about 20×20 pixels, so we have to scaled some random samples to reduce the target to about 20×20 pixels. To ensure the model can learn more small targets, the original image is also scaled by the same scale. The backbone network of YOLOv5 used skip-connection to reduce the number of layers, and added the Feature Pyramid Networks to different layers to extract more features. Finally, we also used cross validation to train the model.

Because there is a large file that cannot be uploaded in github. So we put the files in google drive.

`https://drive.google.com/file/d/193B3I6eMcYHfL_ieWrTYYmTULeuT1Cri/view?usp=sharing`

2.3 Inverse Perspective Transformation

2.3.1 Motivation

Existing attempts on the internet for detecting social distance violation often ignores the fact that distances can be distorted for images captured by camera with a certain height and angle to the ground [7]. Hence, their methods are prone to false positives, especially for pedestrians whose physical distances are far from the camera and therefore appear closer in pixels.

To properly compute distance, one needs to study homography transformation and learn how objects in 3D euclidean space are projected onto 2D image. The transformation is described by the equation below

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = H_{3\times 3} \begin{bmatrix} x^{\text{BEV}} \\ y^{\text{BEV}} \\ 1 \end{bmatrix} \tag{1}$$

where we make an inherent assumption that pedestrians are walking on the same plane sharing the same z^{BEV} coordinate, and x^{BEV}, y^{BEV} denote the 2D euclidean coordinate of the pedestrian on the plane. Notice that the transformation does not necessarily preserves the L2-norm of distances between vectors.

To accurately compute the distances, one needs to obtain the inverse of H to reconstruct the bird eye view coordinates that properly encodes distances.

$$H^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x^{\text{BEV}} \\ y^{\text{BEV}} \\ 1 \end{bmatrix} \tag{2}$$

Notice that the inverse of a homography transformation is also a homography transformation because it is a closed group [6]. As detailed in lecture, construction of H and H^{-1} usually requires information such as camera height and angle.

When these parameters are not available, it is also possible to handpick the transformation of at least four points to determine the form of H^{-1}. This is detailed in the method section.

2.3.2 Homography from Projection

In this section we showed why, with proper assumption, it is possible to use only homography transformations to describe the transformation between 3D coordinates to 2D pixels of the pedestrians.

The most general transformation between 3D world coordinates and 2D pixel coordinates is a 3 by 4 projection matrix. However, using the pedestrians' world coordinates on the ground to indicate where they are and further assuming that the ground is flat, we notice that all points of interest are on the same plane. In this special case, we can describe the transformation of all points of interest using the homography transformation.

As discussed in lecture, the projection matrix P

$$\begin{bmatrix} ax \\ ay \\ a \end{bmatrix} = P_{3\times 4} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3}$$

maps 3D points into 2D points. P can be decomposed as follow

$$P_{3\times 4} = K_{3\times 3} E_{3\times 4} \tag{4}$$

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

$$E = \begin{bmatrix} R_{3\times 3} & T_{3\times 1} \end{bmatrix} \tag{6}$$

$$= \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \end{bmatrix} \tag{7}$$

where f, p_x, p_y are the intrinsic parameters and R, T respectively describes the rotation and translation of the camera in world coordinate.

Making the assumption that all points of interest are on the same plane, we can rotate and translate our coordinate system to find one in which the plane lies on $Z = 0$. Thus, with interesting points on the $Z = 0$ plane, we find the following form of transformation

$$\begin{bmatrix} ax \\ ay \\ a \end{bmatrix} = K \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} \tag{8}$$

It is then evident that R_{13}, R_{23}, R_{33} are unnecessary. And the same transformation can be modeled by

$$\begin{bmatrix} ax \\ ay \\ a \end{bmatrix} = K \begin{bmatrix} R_{11} & R_{12} & T_1 \\ R_{21} & R_{22} & T_2 \\ R_{31} & R_{32} & T_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{9}$$

$$= H_{3\times 3} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{10}$$

Hence, we only need the 3×3 homography transformation to reconstruct the bird-eye view coordinate [8].

In this parametrization, there are 9 parameters from rotation (3), translation (3), and the three internal parameters. This obviously over-parametrizes homography transformation, which in general has 8 degrees of freedom (DOF). Perhaps, the extra DOF comes from the fact that we removed an entire column from the rotation matrix, and one of the angle is not needed.

2.3.3 Alternative parametrization of Homography

According to Hartley & Zisserman (2003), the homography H can be decomposed as follow [6]

$$H = \begin{bmatrix} sR & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} K & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} I & 0 \\ v^T & w \end{bmatrix} \tag{11}$$

where s, w are numbers (2 DOF), R is a $2x2$ rotational matrix with 2 DOF, K is a $2x2$ determinant-1 upper triangular matrix with 2 DOF, and t, v are both two dimensional vectors each with 2 DOF. Therefore, there are in total 8 degrees of freedom, as one would expect for homography.

Particularly, the upper triangular matrix K with $det(K) = 1$ is parametrized as follow

$$K = \begin{bmatrix} a & b \\ 0 & a^{-1} \end{bmatrix} \tag{12}$$

where it is evident that there are 2 DOF.

2.3.4 Additional Structure: Perspectivity

In our case, we do not need all 8 DOF for homography. The homography we need to deal with is a subset of all possible homography, called perspectivity[6]. This arise due to the fact that a camera has a particular perspective, and that physically restricts the form of the the resulting homography matrix.

However, Hartley & Zisserman (2003) did not provide the decomposition of perspectivity in terms of 6 degrees of freedom [6]. They instead provided an accessible figure that shows the qualitative differences between homography and perspectivity.

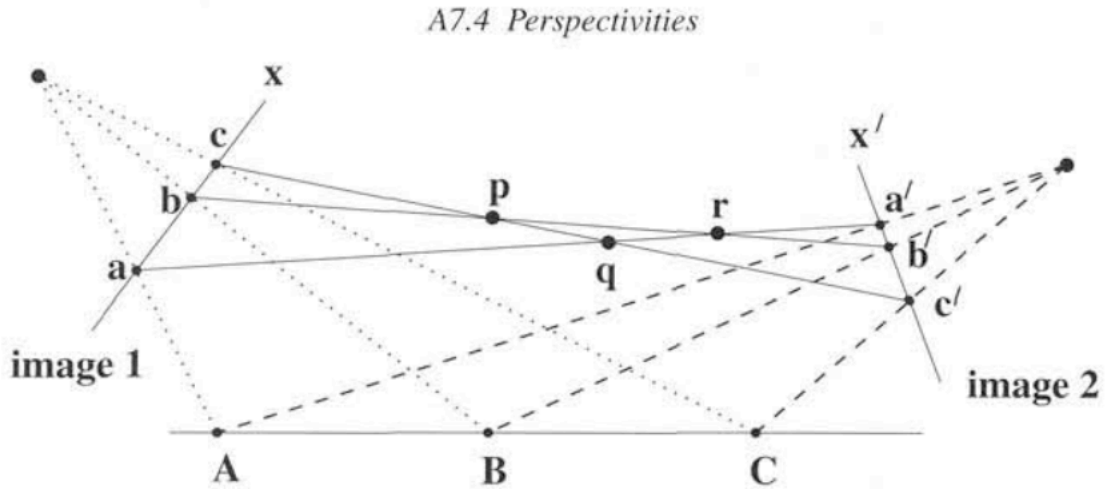


Figure 1: Taken from Hartley & Zisserman (2003)[6].

Here, Images 1 is transformed from the $\overline{ABC}$ plane by a perspectivity. This can be seen from the fact that $\overline{aA}, \overline{bB}$ and $\overline{cC}$ do not overlap, which physically makes sense for a camera looking at the ground at a particular angle. Similarly, Image 2 is transformed from the $\overline{ABC}$ plane by a perspectivity.

However, the homography transformation between Images 1 and 2 is not a perspecitivty, since the lines $\overline{aa'}, \overline{bb'}, \overline{cc'}$ intersect. Clearly, homography and perspectivity are qualitatively differect.

2.3.5 Homography as 2D transformation that preserves colinearity

It is possible to prove that a real, 2D transformation preserves colinearity if and only if it can be represented by a $3x3$ non-singular homography matrix. This turns out to be connected to the Fundamental Theorem of Projective Geometry [9], which states that over real field, for dimension greater than 2, any transformation that preserves colinearity (colineation) is a homography [10].

It is easier to see that homography preserves colinearity, and we presented a variant of the argument by Hartley & Zisserman (2003) here [6].

Consider the equation satisfied by points (x, y) on a line defined by $\begin{bmatrix} a & b & c \end{bmatrix}^T$.

$$ax + by + c = 0 \tag{13}$$

Let

$$\vec{x} = \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} \tag{14}$$

$$\vec{a} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{15}$$

where $s \in (-\infty, \infty)$ is there to account for all points (sx, sy) on the same line. Then, the same equation of a line can be represented as a inner product

$$\vec{a}^T \vec{x} = 0 \tag{16}$$

Consider an invertible homography transformation H.

$$\vec{w} = H\vec{x} \tag{17}$$

$$H^{-1}\vec{w} = \vec{x} \tag{18}$$

$$\vec{a}^T H^{-1} \vec{w} = \vec{a}^T \vec{x} = 0 \tag{19}$$

Then it is evident that

$$\left[\left(H^{-1}\right)^T \vec{a}\right]^T \vec{w} = 0 \tag{20}$$

and the set of transformed points $\vec{w}$ are on a line defined by $\left(H^{-1}\right)^T \vec{a}$. This shows that homography indeed preserves colinearity.

3 Methodology, Results, and Experiments

3.1 Face Mask Detection

3.1.1 Problem Description

Our goal is to identify whether a pedestrian is wearing a face mask once we obtain an image of the pedestrian's. We identify the problem of pedestrian

face mask detection as a binary classification problem, that is, we look to classify an image into one of two classes; masked or unmasked. Given that in our application, face mask detection will likely be performed on low resolution images captured from security cameras, it is important that our model accurately classify masked and unmasked pedestrians under this constraint.

3.1.2 Model Architecture

We use a Convolutional Neural Network to classify whether an image contains the face of masked individual or an unmasked individual. We decide to use a CNN in our approach because it allows to perform both feature learning; a process by which our model learns image features and creates a flattened feature vector, and classification. The input to our CNN is a 16x16 grayscale image, that is, our CNN has one input channel. We downscale images to 16x16 to mimic low resolution security camera footage. The feature learning section in our model contains two layers each of which include a convolution step and a pooling step.

We perform a convolution step in each layer using a convolution window of size 3x3 and a window stride of one pixel. To ensure that the dimensions of our input image are preserved after the convolution step we apply zero padding on each image dimension. We choose to preserve image dimension to prevent shrinking of our input image which we assume to have low resolution.

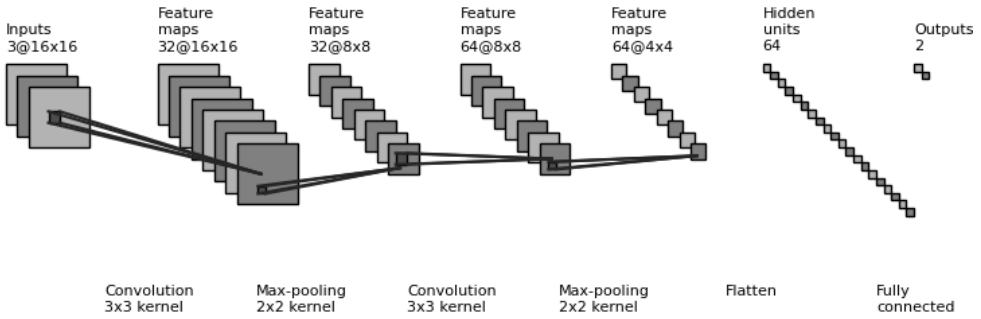
After each convolution step, we perform a max pooling operation using a 2x2 kernel to perform dimensionality reduction and noise suppression [11]. We decide to use a max pooling layer to down-sample our image by extracting the local maximum values from each 2x2 image section. The reduced dimensionality lessens the total number of parameters included in the feature vector passed to the classification network, thus, decreasing the overall computational load required for training [11].

Additionally, after each layer in our CNN we perform a rectified linear activation (ReLU). A consequence of performing convolution is the introduction of linearity within the image, that is, the gradual change of colour between pixels [12]. To reintroduce non-linearity in the image and scale

any negative pixel values to zero we use a ReLU activation function [12].

$$ReLU(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

After feature extraction we obtain a 1x1024 flattened feature vector which we pass to our first and only fully connected layer for classification. Our fully connected layer has an input dimension of 1024 and an output dimension of 64. In our output layer we use a softmax activation function which assigns a probability value to the masked and unmasked classes, where the higher probability class is deemed the prediction by the model. We present the forward pass of our model;



3.1.3 Dataset

The selection of our dataset is an important factor to the accuracy of our results. The Face Mask Detection dataset [13] contains 3835 images of which 1916 images are of individuals wearing a mask and 1919 images are of individuals not wearing a mask. The images in the dataset are a collection of images from the Bing Search API, Kaggle datasets and RMFRD (Real-World Masked Face Dataset). A reason as to why we chose the Face Mask Detection dataset is because it consists of images from Kaggle datasets [14] and RMFRD [15] which are also used in *Multi-Stage CNN Architecture for Face Mask Detection* [16] to accomplish a similar task. We provide sample images from the dataset;

Figure 2: Sample images from the Face Mask Detection dataset.

The Face Mask Detection dataset is pre-labeled into the masked and unmasked classes and a majority of images in the dataset consist of individuals wearing real face masks as opposed to an edited face mask overlay. We provide examples of edited face mask overlays;

Figure 3: Examples of edited face mask overlays

Since there are images scraped from the Bing Search API, there exists uncharacteristic images of face masks including images containing more than one individual, video thumbnails and mask advertisements. Note that these uncharacteristic comprise a small subset of the entire dataset and so, we have not removed these samples.

Figure 4: Uncharacteristic sample images in the Face Mask Detection dataset.

3.1.4 Model Parameter Tuning & Analysis

In our model, we attempt to minimize Cross Entropy loss using the Adaptive Moment Estimation optimizer [17]. We have chosen Adaptive Moment Estimation optimizer since it dynamically scales the learning rate using first and second moment estimation and it uses the moving average of the gradient in weight decay as opposed to the gradient itself [17]. Primarily, the Adaptive Moment Estimation optimizer speeds up training as compared to other optimizers such as Stochastic Gradient Descent. We first provide an analysis of the tuned learning rate for our model.

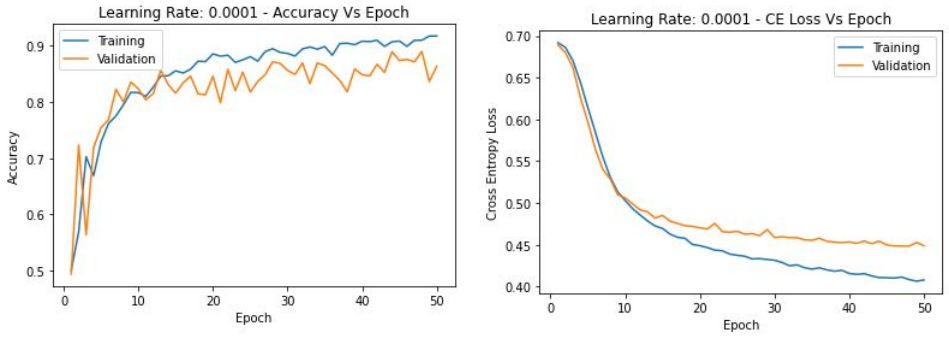


Figure 5: Learning Rate: 0.0001 | Epochs: 50 | Batch Size: 32

Accuracy on validation dataset is 0.8660869598388672

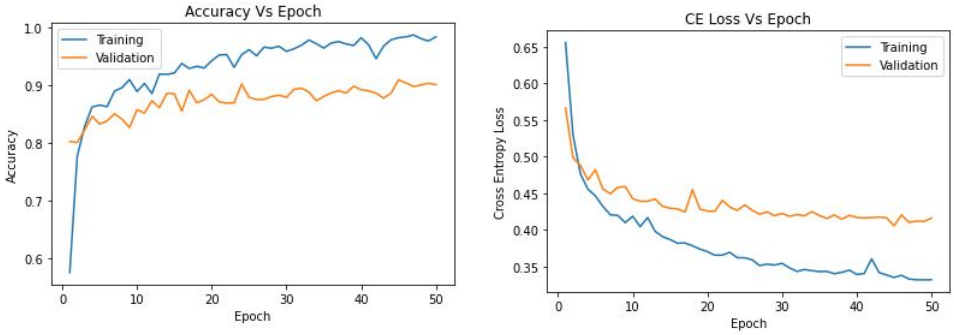


Figure 6: Learning Rate: 0.001 | Epochs: 50 | Batch Size: 32

Accuracy on validation dataset is 0.895652174949646

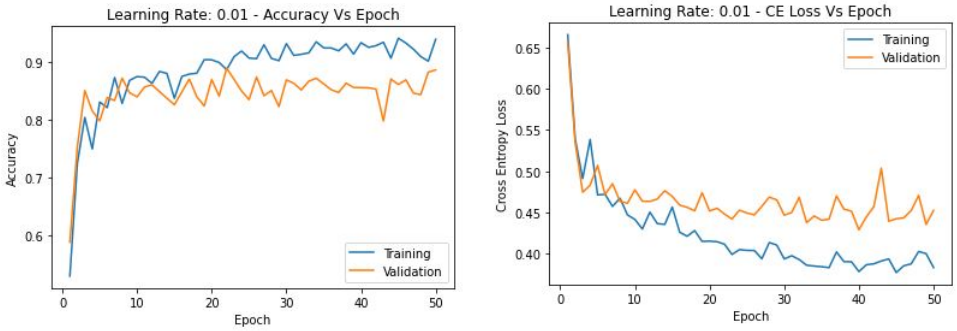


Figure 7: Learning Rate: 0.01 | Epochs: 50 | Batch Size: 32

Accuracy on validation dataset is 0.8573912978172302

Given the above Accuracy VS. Epoch plots, we obtain the highest validation accuracy of 0.895 (Figure 6) when using a learning rate of 0.001. We observe potential overfitting in Figure 6 as neither training or validation accuracies and Cross Entropy losses converge, which suggests that we must experiment with the number of epochs and or the batch size. Next, we analyze the effects of batch size on the accuracy and cross entropy loss of our model.

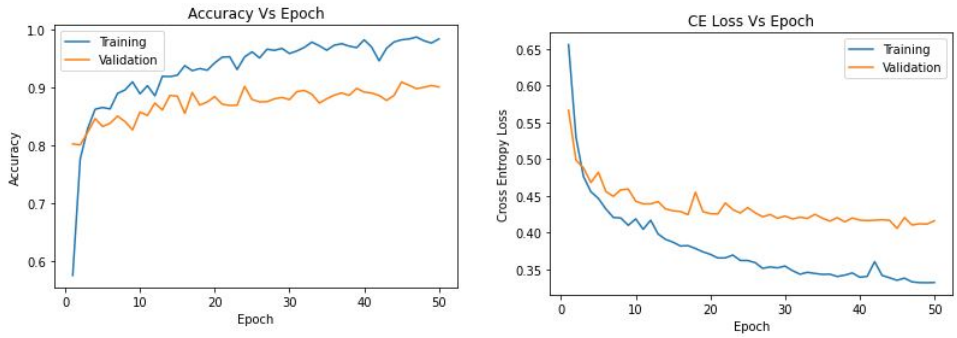


Figure 8: Learning Rate: 0.001 | Epochs: 50 | Batch Size: 32

Accuracy on validation dataset is 0.895652174949646

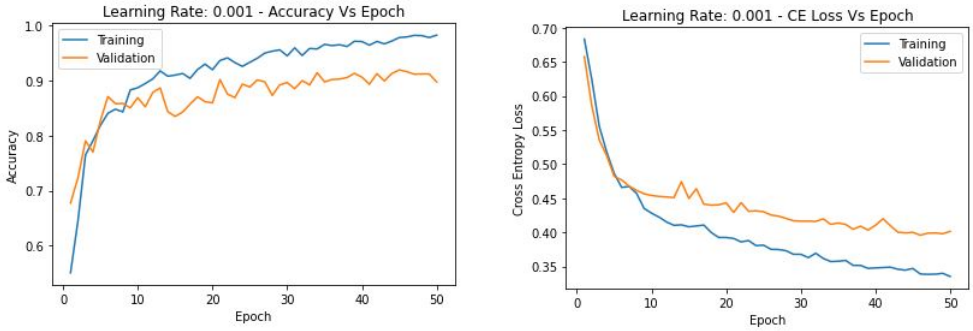


Figure 9: Learning Rate: 0.001 | Epochs: 50 | Batch Size: 64

Accuracy on validation dataset is 0.8991304636001587

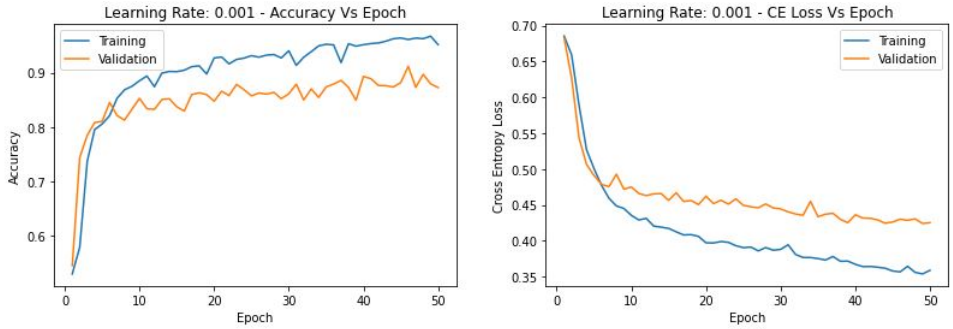


Figure 10: Learning Rate: 0.001 | Epochs: 50 | Batch Size: 128

Accuracy on validation dataset is 0.8799999952316284

We choose a batch size of 64 since it gives us the highest validation accuracy of 0.899 (Figure 9). We note that as the number of epochs increases the variation in the validation accuracy tends to decrease, therefore we experiment larger number of epochs. We obtain the best validation accuracy using 500 training epochs as presented below;

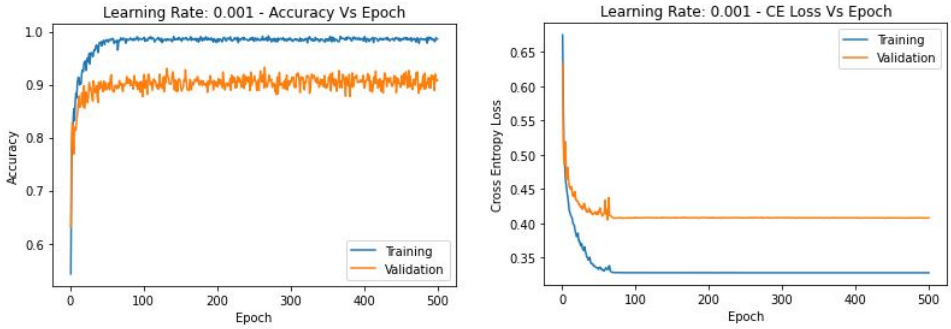


Figure 11: Learning Rate: 0.001 | Epochs: 500 | Batch Size: 64

Accuracy on train dataset is **0.985**
Accuracy on validation dataset is **0.904**
Accuracy on test dataset is **0.901**

The tuned parameters we use for our model are a learning rate of 0.001, 500 training epochs and a batch size of 64 images.

3.1.5 Results Analysis

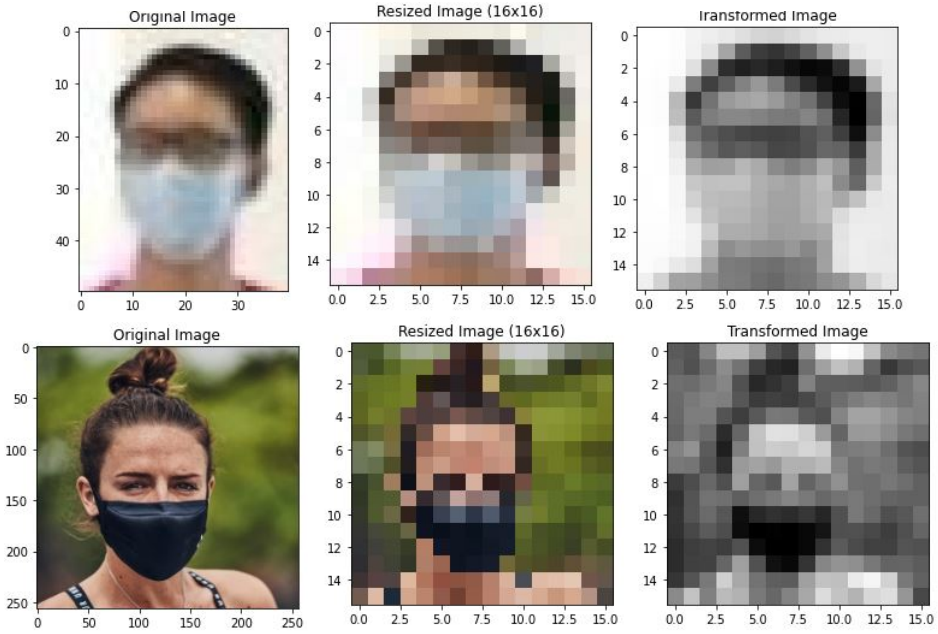


Figure 12: Examples of correctly classified individuals wearing a mask.

Figure 13: Examples of correctly classified individuals not wearing a mask.

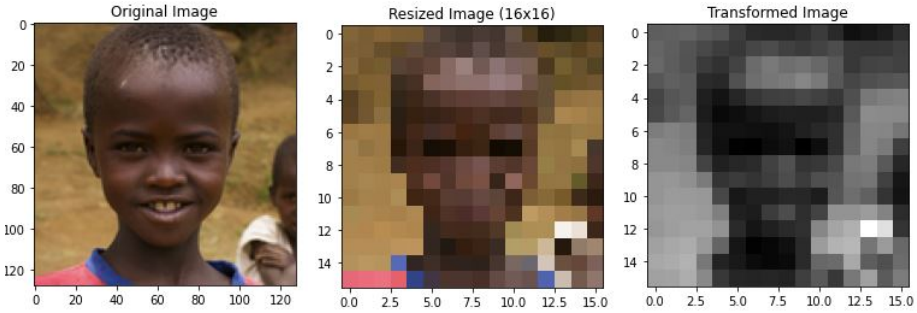


Figure 14: Example of misclassified individual

The misclassification in Figure 14 arises from a bias in the dataset of individuals from specific regions of the world and the lack of representation of people from other regions, as reported in *Multi-Stage CNN Architecture for Face Mask Detection* [16].

Please view facemask_detect_low_res.ipynb for a visual representation of more classified examples.

3.2 Compute interpersonal distances

3.2.1 Problem Description

Given the bounding boxes enclosing pedestrians, we wish to apply homography transformation and compute the interpersonal distances in the world coordinate. Additionally, for the few pedestrians who do not wear masks, we require twice the distances from others. For those violating the social distancing measure, we color the boxes in red.

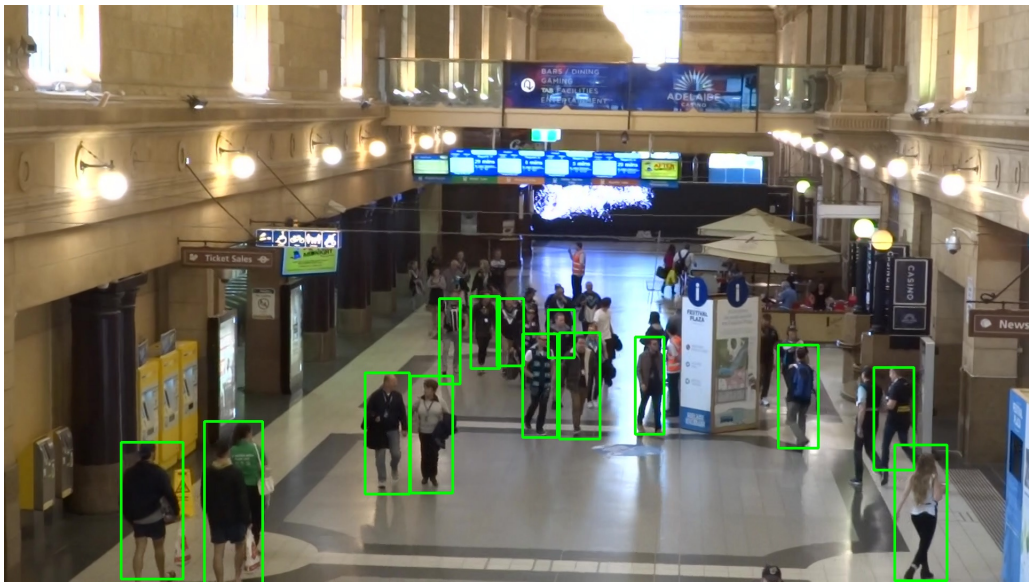


Figure 15: The output from previous sections.

3.2.2 Homography

While we assumed that internal and external parameters of the camera is known and therefore H is given, in practice, we do not have this knowledge. We instead constructed homography by choosing four points on the floor and map them to the corners of an rectangle. Using materials from the

lecture, we computed homography H from constructing the matrix A

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i \\ & & & & \cdots & & & & \end{bmatrix} \tag{21}$$

$$h = \begin{bmatrix} h_1 \\ h_2 \\ \cdots \end{bmatrix} \tag{22}$$

where h_i are the coefficients of homography matrix, and the primed coordinates are the transformed coordinates. In this formulation, we want

$$Ah = 0 \tag{23}$$

Of course, this is not generally possible and it will depend on the coordinates (x_i, y_i) and (x_i', y_i') provided. So one instead looks for

$$\min ||Ah||_2^2 \tag{24}$$

The solution h that minimizes the L2 norm of Ah turns out to be the eigenvector of A^TA with lowest eigenvalue.

We implemented this function, applied it on four handpicked points and their transformed coordinates, and compared its output to the normalized output of cv2.FindHomography.

Listing 1: The output of the test

```
Normalized CV2:
[[-1.82768385e-04 -2.80055125e-04  2.85959615e-01]
 [ 2.91483722e-05 -1.30056206e-03  9.58240727e-01]
 [ 1.34396719e-08 -4.18780353e-07  6.19651251e-05]]
Monroe-Penrose:
[[-1.82830507e-04 -2.79261276e-04  2.85221668e-01]
 [ 2.91808376e-05 -1.30089822e-03  9.58460636e-01]
 [ 1.34726204e-08 -4.18096286e-07  6.11426335e-05]]
```

As shown above, the implemented function has roughly the same result as that of cv2.

3.2.3 Parametrization of Homography

Sometimes one may not be interested in simply minimizing the L2-norm of Ah. Moreover, constructing A^TA can be costly depending on the number of points transformed. In these cases, to minimize a given cost function, we need to parametrize h and find the solution variationally. We coded two parametrizations detailed earlier— one constructed the 9-parameters $3x4$ projection matrix and cut it down to $3x3$ and the other implemented the 8-parameters homography.

We then take the homography matrix H obtained earlier, and tested the two parametried homography H_p using scipy.optimize.minimize on

$$\min ||H - H_p||_2^2 \tag{25}$$

to see whether the optimizations converge. It counts as a successfully converged optimization when the L2-norm is below threshold 10^{-4}.

Running 10 optimizations with randomly initialized parameters, we showed the rate of success convergences below

Listing 2: The output of the optimizations. Lecture form is the parametrization with 9 degrees of freedom. And SAP form is the more efficient parametrization with 8 degrees of freedom.

```
Converged L2 norm. Lecture form: 1.2849988460865852e-06
Converged L2 norm. SAP form: 0.0001541852243421205
Percentage of success parametrization using lecture form: 1.0
Percentage of success parametrization using SAP form: 0.1
```

Notice that using the efficient parametrization results in a lower success rate. This is not surprising, since the overparametrized model is expected to have an easier time to converge.

3.2.4 Results

After obtaining the homography transformation, we then choose the middle-bottom coordinates of the bounding boxes to represent the position of pedestrians. This choice allows us to use the homography obtained from the transformation of the floor pixels.

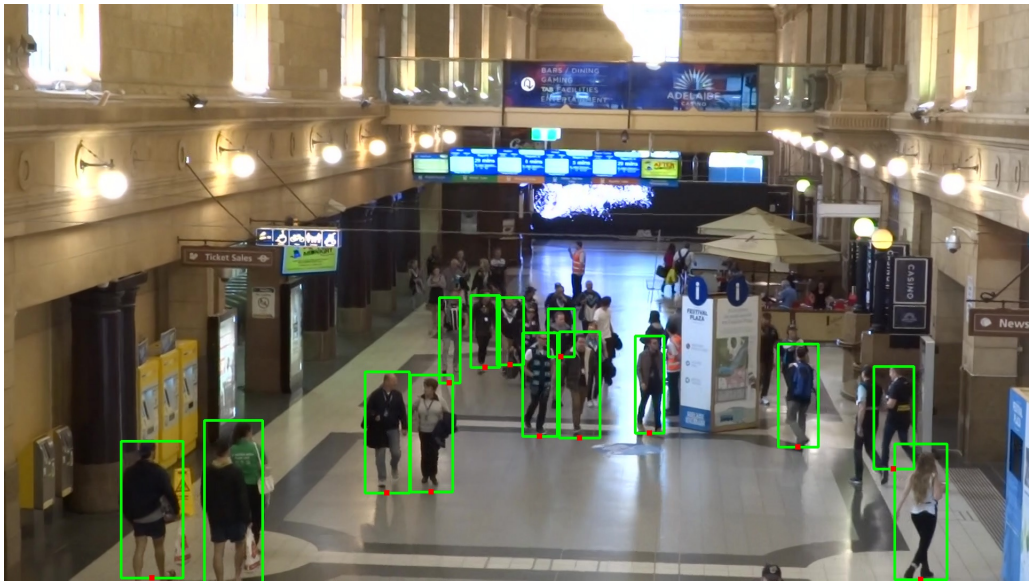


Figure 16: The coordinates representing the pedestrians is colored in red.

Then, it is straightforward to apply it and compute the eucliean distances in world coordinates and find the coordinates that are too close. We then colored the corresponding boxes in red.

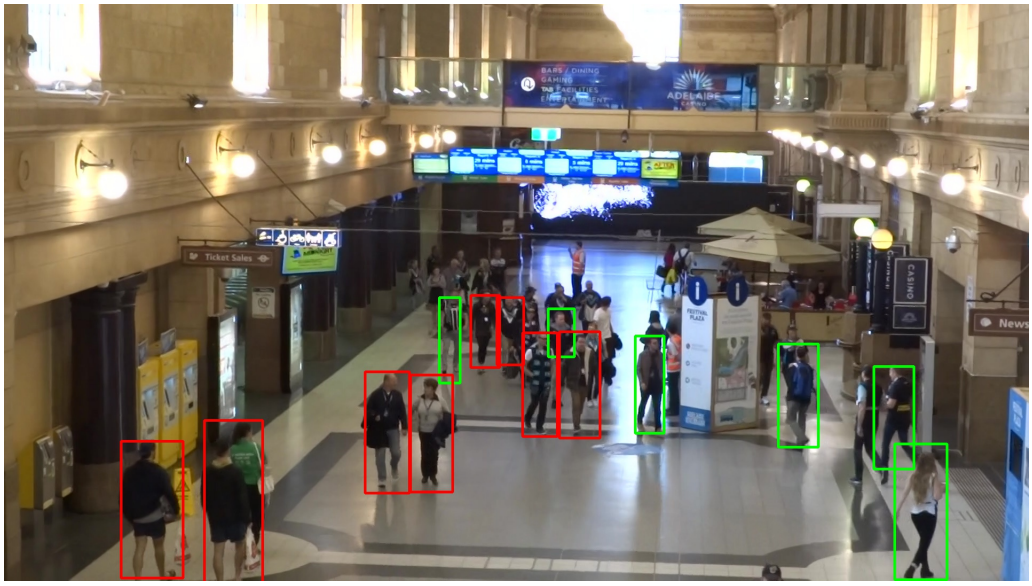


Figure 17: The pedestrians that are too close to each other are in red bounding boxes.

One can see that the algorithm generally works well, but there are two interesting failed cases.

First, the leftmost green box does not reflect where the person is — it is a bit to the left of where the person is standing. Therefore, the algorithm does not see that the person is also very close to the other two people on the right. Second, the person at the middle is covered by two people in front, so the bounding box fail to extend all the way to the floor. Consequently, our program could not tell how close is this person to the others.

Additionally, we tested the functionality of requiring pedestrians without mask to have greater physical distances with others. Assuming that the pedestrian in green box at the middle is the one without mask, we applied the algorithm again.

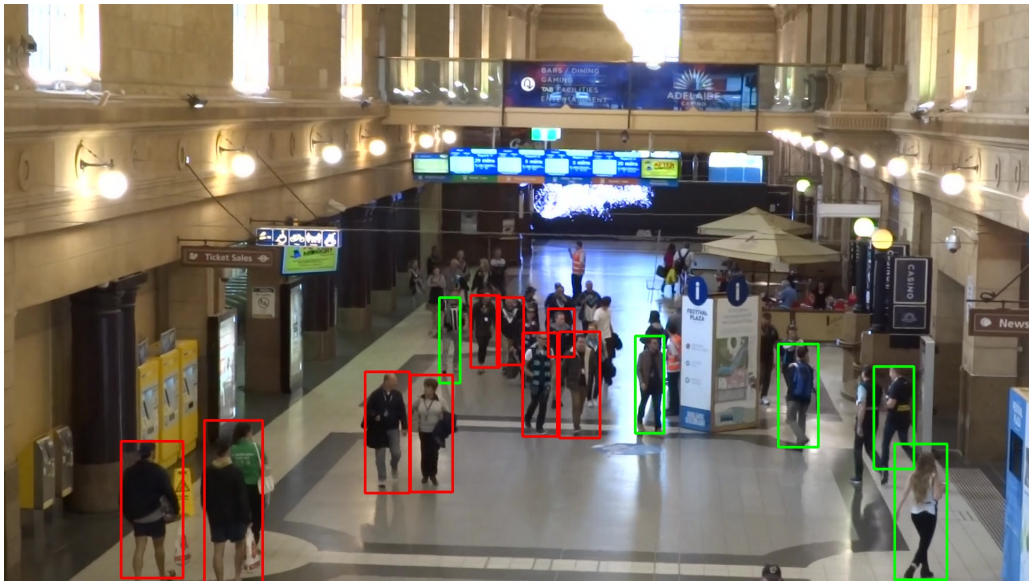


Figure 18: Require doubled physical distances from the pedestrians at the middle.

As expected, the pedestrian is now in a red bounding box with a stricter social distancing measure.

4 Conclusion

This project built a software that detects pedestrians who violate the social distancing measure. In addition to the usual practices, we further identify individuals who wear masks and require a longer physical distances from them. We also explored various properties of homography transformation, including its parametrization.

There are some missing detections that arise due to inaccurate placements of the bounding boxes. Hence, for future work, one may attempt to train the pre-trained YOLO model to a dataset that is more accurate in the placement of bounding boxes over pedestrians. Moreover, judging from the pixel coordinates and relative sizes of bounding boxes, one may be able to develop post-processing algorithm that corrects the height of the bounding boxes despite obstacles.

Moreover, due to the low resolution images of pedestrians within a scene,

the face detection given a pedestrian is inaccurate. In a future application, we can look to use pose-detection to estimate the location of an individual's face, since it is independent of the image quality and is in reference to an individual's body. As noted in section 3.1.5 there are misclassified examples of individuals who are classified as wearing a mask. These misclassifications can potentially arise from regional bias that exists in the dataset. Also, the Face Mask Detection dataset contains uncharacteristic examples of individuals wearing a face mask which could also affect the accuracy of our model. In a future application, it is important that we remove any uncharacteristic examples and diversify the dataset with people of different regional backgrounds to make our model more robust and accurate.

5 Author's Contributions

Tzu-Ching Yen

- Write the abstract, introduction.
- Write the conclusion with Muneeb.
- Research ways to obtain bird eye view coordinates of pixels.
- Research properties and parametrizations of homography and perspectivity.
- Prepare the outputs of YOLOv3 for social distancing detection.
- Apply, test, and present the performance of social distancing detection.

Muneeb Ansari

- Implement pedestrian detection using pretrained Darknet model from YOLOv3.
- Use Dlib library method to perform face detection on an image
- Research conovolutional neural networks
- Build face mask detection model using PyTorch

- Tune face mask detection model parameters and present analysis
- Research different face mask datasets
- Write the conclusion with Tzu-Ching.

Gang Zhao

- Research on YOLOv3, YOLOv4
- Train a pedestrian detection model

References

1. Siedner, M. J. *et al.* Social distancing to slow the US COVID-19 epidemic: Longitudinal pretest–posttest comparison group study. *PLOS Medicine* **17,** 1–12 (2020).
2. Yang, D. *et al.* A Vision-based Social Distancing and Critical Density Detection System for COVID-19, arXiv:2007.03578 (2020).
3. Cristani, M. *et al.* The Visual Social Distancing Problem, arXiv:2005.04813 (2020).
4. 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:2005.01385 (2020).
5. Redmon, J. & Farhadi, A. YOLOv3: An Incremental Improvement, arXiv:1804.02767 (2018).
6. Hartley, R. & Zisserman, A. *Multiple View Geometry in Computer Vision* 2nd ed. ISBN: 0521540518 (Cambridge University Press, 2003).
7. MK, G. *COVID-19: Face Mask Detection using TensorFlow and OpenCV* https://towardsdatascience.com/covid-19-ai-enabled-social-distancing-detector-using-opencv-ea2abd827d34.
8. Ramanan, D. *Computer Vision* http://16720.courses.cs.cmu.edu/.
9. Faure, C.-A. An Elementary Proof of the Fundamental Theorem of Projective Geometry (Dedicated to Alfred Frölicher). *Geometriae Dedicata* **90,** 145–151 (2002).

10. Hirschfeld, J. W. P. *Projective geometries over finite fields* ISBN: 0198502958 (Clarendon Press, 1979).
11. Saha, S. *A Comprehensive Guide to Convolutional Neural Networks-the ELI5 way* *https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53* Dec. 2018.
12. *Convolutional Neural Networks (CNN): Step 1(b) - ReLU Layer* *https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-1b-relu-layer/*
13. chandrikadeb7. *Face Mask Detection Dataset* *https://github.com/chandrikadeb7/Face-Mask-Detection*
14. Larxel. *Face Mask Detection* May 2020.
15. Wang, Z. *et al. Masked Face Recognition Dataset and Application* 2020.
16. Chavda, A., Dsouza, J., Badgujar, S. & Damani, A. *Multi-Stage CNN Architecture for Face Mask Detection* 2020.
17. Bushaev, V. *Adam-latest trends in deep learning optimization.* *https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c* Oct. 2018.