Kaunas University of Technology

Faculty of Informatics

Investigation of Social Distancing Monitoring Using Multi-View Detectors

Master's Final Degree Project

Rokas Janavičius

Project author

Doc. Dr. Armantas Ostreika

Supervisor

Kaunas, 2024

Kaunas University of Technology

Faculty of Informatics

Investigation of Social Distancing Monitoring Using Multi-View Detectors

Master's Final Degree Project

Artificial Intelligence in Computer Science (6211BX007)

Rokas Janavičius

Project author

Doc. Dr. Armantas Ostreika

Supervisor

Doc. Dr. Gintaras Palubeckis

Reviewer

Kaunas, 2024

Kaunas University of Technology

Faculty of Informatics

Rokas Janavičius

Investigation of Social Distancing Monitoring Using Multi-View Detectors

Declaration of Academic Integrity

I confirm that the final project of mine, Rokas Janavičius, on the topic „Investigation of Social Distancing Monitoring Using Multi-View Detectors“ is written completely by myself; all the provided data and research results are correct and have been obtained honestly. None of the parts of this thesis have been plagiarised from any printed, Internet-based or otherwise recorded sources. All direct and indirect quotations from external resources are indicated in the list of references. No monetary funds (unless required by Law) have been paid to anyone for any contribution to this project.

I fully and completely understand that any discovery of any manifestations/case/facts of dishonesty inevitably results in me incurring a penalty according to the procedure(s) effective at Kaunas University of Technology.

(name and surname filled in by hand)	(signature)

Kaunas University of Technology

Faculty of Informatics

Task of the Master's final degree project

Topic of the project	Investigation of Social Distancing Monitoring Using Multi-View Detectors
Requirements and conditions (title can be clarified, if needed)	
Supervisor	
	(position, name, surname, signature of the supervisor) (date)

Janavičius, Rokas. Investigation of Social Distancing Monitoring Using Multi-View Detectors. Master's Final Degree Project. Supervisor doc. dr. Armantas Ostreika; Faculty of Informatics, Kaunas University of Technology.

Study field and area (study field group): Computer science, Informatics (B01).

Keywords: computer vision, social distancing, multi-view detector.

Kaunas, 2024. 65 pages.

Summary

Observing social distance is essential in maintaining personal health and safety, as it is one of the essential preventive measures against airborne viruses and diseases. Therefore, research is constantly being carried out to create technological solutions to ensure accurate social distance detection in public spaces. The main existing security methods are based on single-camera images analysed by convolutional neural networks. These models identify the people in the frame and transform the camera view into a "top-down" perspective, allowing the space of each detected person and the distance to the surroundings to be determined.

The main goal of this study is to apply a multi-view pedestrian detection algorithm to a social distance monitoring system. Furthermore, the impact of the spatial placement of cameras on tracking accuracy and other important detection indicators is also investigated. In addition, existing social distance monitoring solutions and their algorithms are reviewed.

The *EarlyBird* algorithm chosen for the research uses images from several cameras, extracting their essential features, performing perspective transformation, and combining the obtained results to determine the most likely positions of individuals. Improvements made during the study allow a straightforward selection of available cameras for evaluating the quality of social distance monitoring. Finally, potential improvements to the initial stage of the algorithm are explored using more advanced convolutional neural network models for person detection.

This research demonstrates that a multi-view pedestrian detection system significantly improves accuracy in detecting social distancing violations. Moreover, utilising a more complex backbone for feature map extraction did not substantially enhance accuracy. However, using the less complex *TinyNet-E* model resulted in faster training and inference times, with only a marginal reduction in accuracy.

Janavičius, Rokas. Socialinio atstumo stebėjimo naudojant daugiavaizdžius detektorius tyrimas. Magistro baigiamasis projektas. Vadovas doc. dr. Armantas Ostreika; Kauno technologijos universitetas, informatikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Informatikos mokslai, Informatika (B01).

Reikšminiai žodžiai: kompiuterinė rega, socialinis atstumas, daugiavaizdis detektorius.

Kaunas, 2024. 65 p.

Santrauka

Socialinio atstumo stebėjimas yra svarbus uždavinys asmens sveikatos saugumo sferoje, nes socialinio atstumo laikymasis yra viena iš esminių prevencinių priemonių prieš oru plintančius virusus ir ligas. Todėl nuolat vykdomi tyrimai siekiant sukurti technologinius sprendimus, kurie užtikrintų tikslų socialinio atstumo aptikimą viešose erdvėse. Pagrindiniai egzistuojantys metodai remiasi pavienių apsaugos kamerų vaizdais, kuriuos analizuoja įvairūs konvoliuciniai neuroniniai tinklai. Šie modeliai identifikuoja kadre esančius žmones ir transformuoja kameros vaizdą į „iš viršaus į apačią“ perspektyvą, leidžiančią nustatyti kiekvieno aptikto asmens padėtį erdvėje bei atstumą iki aplinkinių.

Šio tyrimo pagrindinis tikslas - ištirti daugiavaizdžio asmenų aptikimo algoritmo pritaikymą socialinio atstumo stebėjimui. Taip pat tiriamas kamerų išdėstymo erdvėje ir jų skaičiaus poveikis stebėjimo tikslumui bei kitiems svarbiems aptikimo rodikliams. Papildomai yra apžvelgiami esami socialinio atstumo stebėjimo sprendimai, jų naudojami algoritmai ir pritaikymai skirtingose viešosiose erdvėse.

Tyrimui vykdyti pasirinktas algoritmas *EarlyBird* naudoja kelių kamerų vaizdus, jų esminių savybių išgavimui, atlieka perspektyvos transformaciją ir sujungia gautus rezultatus, siekiant nustatyti labiausiai tikėtinas asmenų pozicijas. Tyrimo metu atlikti patobulinimai leidžia lengvai pasirinkti bet kokius galimus kamerų kampus ir jų skaičių matavimams vykdyti, taip pat įvertinti socialinio atstumo stebėjimo kokybę. Galiausiai, tiriami potencialūs patobulinimai pradinėje algoritmo stadijoje, naudojant pažangesnį asmenų aptikimo konvoliucinių neuroninių tinklų modelį.

Tyrimo rezultatai rodo, kad daugiavaizdžio asmenų aptikimo sistema ženkliai pagerina socialinio atstumo pažeidimų aptikimo tikslumą. Taip pat aptikta, kad sudėtingesnio konvoliucinio tinklo naudojimas esminių savybių išgavimui nežymiai įtakoja galutinį sistemos tikslumą. Tačiau naudojant paprastesnį *TinyNet-E* modelį tam pačiam tikslui, asmenų aptikimo trukmė pastebimai sutrumpėja, o tikslumas pakinta minimaliai.

Table of contents

List of figures

List of tables

List of abbreviations and terms

Abbreviations:

ANN – Artificial Neural Network

CNN – Convolutional Neural Network

WHO – World Health Organisation

CSV – Comma Separated Values

FPT – Feature Perspective Transform

CUDA - Compute Unified Device Architecture

MODA – Multiple Object Detection Accuracy

MODP – Multiple Object Detection Precision

TP – True Positive

TN – True Negative

FP – False Positive

FN – False Negative

Terms:

Tfevents file – a file format used by the TensorFlow library to store machine learning model training and evaluation metrics.

Feature map – a multi-dimensional array containing the results of a convolutional operation.

Heatmap – a graph containing data values represented by colours

Introduction

Project novelty and relevance

Social distancing monitoring has become crucial to protecting public health, especially with the recent COVID-19 pandemic. Maintaining space between people is a key preventative measure against airborne viruses. By detecting social distancing violations in crowded areas, we gain valuable insights that can inform targeted decisions regarding crowd management, object placement, and planning.

The analysis of existing methods revealed their main limitations, particularly in handling occlusions. To address these issues, a novel multi-view-based social distancing detection system is proposed. It leverages multiple camera feeds to more accurately predict social distancing violations by reducing the negative effects of occlusions. Furthermore, multi-view detector improvements are proposed, focusing on efficiency and accuracy.

Aim and objectives

The main goal of this research is to analyse existing social distancing detection systems and propose a solution for improving the accuracy and violation detection rate of these systems. To achieve this goal, the following objectives were set:

1. Review existing social distancing detection systems and algorithms, identifying fundamental limitations.
2. Create a novel multi-view social distancing detection system that can be evaluated and compared against single-view systems.
3. Evaluate the implemented system on multiple datasets and compare the social distancing detection accuracy.
4. Investigate and implement changes to enhance the accuracy of the multi-view detection system.
5. Explore and implement improvements to the efficiency of the multi-view detection system.

Document structure

This document outlines the development and evaluation of a multi-view social distancing detection system. The analysis section examines current methods and technologies used for social distancing monitoring. The system specifications chapter details the specifics of the proposed system, including its intended uses, requirements, and testing plan. Furthermore, the implementation and evaluation section provides the steps taken to develop the proposed system and overviews the results of testing it, focusing on its effectiveness and potential improvements. Lastly, the key findings and outcomes of the project are summarised in the conclusions.

1. Social Distancing Detection Methods and Algorithms Analysis

This chapter overviews the social distancing detection techniques and algorithms for improving their accuracy. It also analyses convolution methods, together with feedforward and convolutional neural networks. Homography principles and probabilistic occupancy maps are also reviewed. Finally, this chapter outlines the requirements and tools for a multi-view social distancing detection system.

1.1. Existing Solutions

A small selection of existing social distancing detection solutions exists, using a single camera feed to capture a populated area. Afterwards, computer vision principles are applied, and a region-based convolutional network is utilised. It detects individuals and tracks their movement through the area, [1, 2].

Mathematical algorithms are used for person position estimation in space, which estimate the camera distortion and position from its angle, height, and a reference object with known dimensions. Homography principles are applied for this application, where the reference object in some solutions is the average human height and width. After calculating the camera distortion, the feed is transformed to a top-down view. Each of the individual centroid coordinates in space are estimated, and the Euclidean distance between them is calculated. Finally, the event is highlighted and captured in a log if the distance exceeds the configured threshold, [1, 2]. An example of the working solution is displayed in Fig 1.

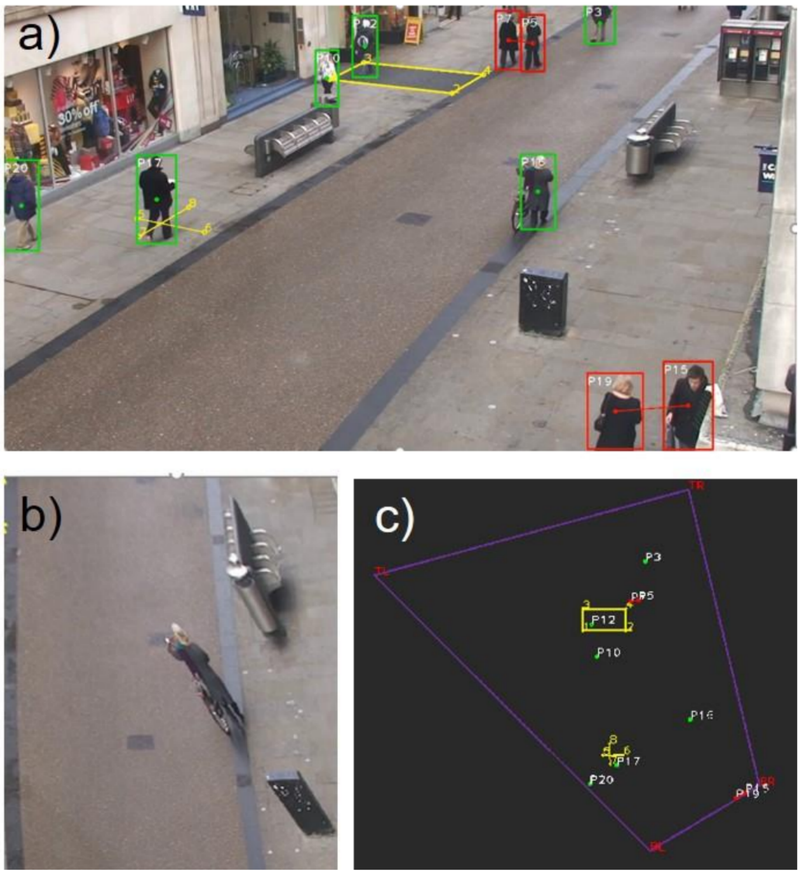


Fig 1. Social distancing detection system solution. a) Detection algorithm with bounding boxes. b) Transformed camera feed to a top-down view. c) Detected individuals with their location in space from a bird's eye view [R1]

1.2. Convolution

Convolution is a mathematical operation combining two functions to produce a new function. It is expressed by the overlapping of one function being shifted over another integral and is widely used in statistics, digital signal processing and image processing. However, depending on the application, a one-dimensional or a two-dimensional convolution operation can be used. They differ from one another mainly by their signal matrix shape. Regardless of the shape, one of the functions is usually regarded as a filter, which can be applied to another function or an image by convolving them both, [3, 4, 5].

The properties of a convolution operations are (where *f* = 1st function, *g* = 2nd function, *h* = function from convolving *f* and *h*), [4]:

- Commutativity property: $f * g = g * f$
- Associative property: $f * (g * h) = (f * g) * h$
- Distributive property: $f * (g * h) = (f * g) + (f * h)$

As convolution is usually used with odd shape filters or kernels, padding must be added around the perimeter of the input matrix. This padding is generally zero-padding, which adds additional zeroes around the perimeter. However, sometimes a more sophisticated technique is used, which averages a specific count of elements and calculates the padded element value, [4, 5].

1.2.1. 1D Convolution

A single-dimensional convolution operation means that both function sizes are a vector. Thus, these functions are usually described by a matrix with one row and an *n* count of numbers in that row. By shifting a filter function over the input one, the integral of their intersection is calculated at each time *t* point. These values are the main target of the convolution operation. The equation (1.1) calculates the single-dimensional convolution operation, where *f* is the 1st function, *g* is the 2nd function, and *t* is the time step for shifting a single function window over the other. An example of a simple convolutional operation is provided in Fig 2, [3, 4, 5].

$$(f * g)(t) := \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau \tag{1.1}$$

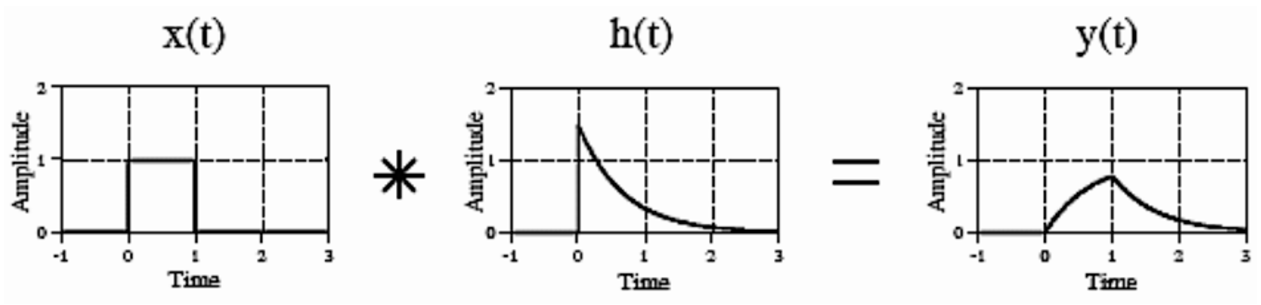


Fig 2. Example of a 1D convolutional operation [R2]

1.2.2. 2D Convolution

A two-dimensional convolution operation is performed on matrices with a shape of N x M. It involves taking a small size filter (e.g., 3x3, 5x5 matrix) with specific values in each element, i.e., weights and sliding it over each input matrix element. Afterwards, each element is multiplied elementwise, and each value is summed up to form a new output element. The resulting matrix is the output of the convolution operation and is always smaller in dimensions compared to the input matrix. The equation (1.2) provides the formula for a two-dimensional convolution operation, where x is the input matrix, and h is the filter matrix. A diagram of a 2D convolution operation is displayed in Fig 3, [4, 6].

$$y[m,n] = x[m,n] * h[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i,j] \cdot h[m-i, n-j] \qquad (1.2)$$

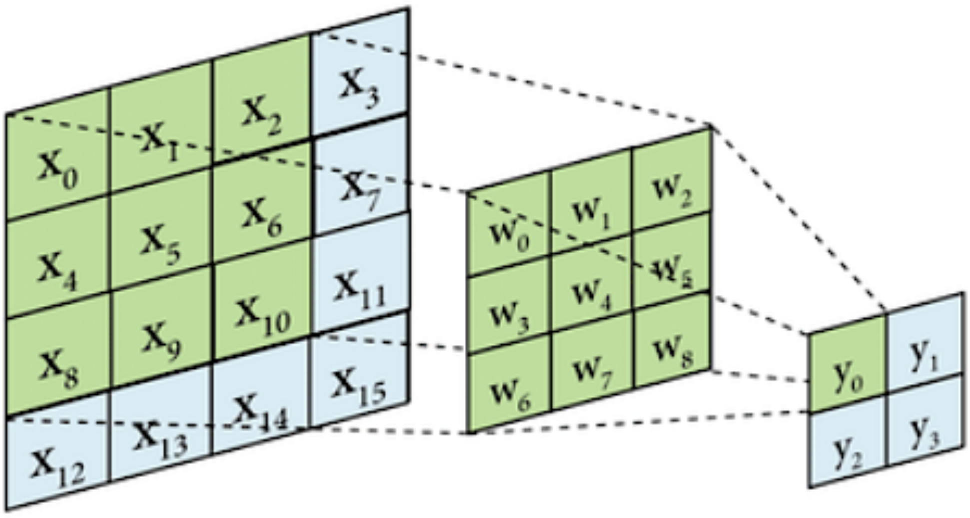


Fig 3. Diagram of a 2D convolution operation [R3]

The filters are also called kernels, and the result can vary greatly depending on their weights. As this type of operation is often utilised in image processing, many specific kernels have been discovered that affect and process an image in a particular way. For example, there are kernels which detect and highlight edges or corners. They are used in edge detection algorithms and machine learning applications. Various blur functions also exist, which blur the image depending on the kernel weights and dimensions. 2D convolution can also be used for image sharpening and masking. An example of a convolution operation used for edge detection and its result is provided in Fig 4, [4, 6].

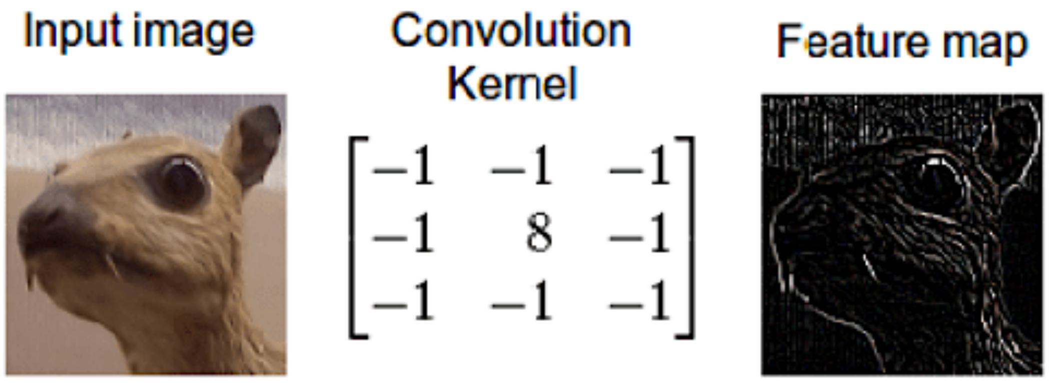


Fig 4. Example of an edge detection convolution operation and its resulting output [R4]

2D convolution operation can be very computationally expensive, as it processes each matrix element and performs many multiplication operations. For example, it can take a long time to compute the convolution when using a large matrix for the input matrix, such as a high-resolution image. This is one of the reasons why machine learning techniques, which use convolution, require better and more expensive computing hardware, [4, 6].

1.3. Homographic Transformation

Homographic transformation is an image processing technique that involves transforming an image to be mapped to another planar projection. This is achieved by selecting at least four known point coordinates and warping the image so these points would appear isometric. This image transformation technique helps shift the camera perspective to a top-down view that can later be exploited for behavioural prediction and autonomous driving, [7, 8].

Homography can also be used for camera pose estimation, perspective removal or correction and panorama stitching. A simple homographic camera perspective change is illustrated in Fig 5, where one camera is the actual camera that captured the image and another is the virtual camera, whose planar view is calculated using homography algorithms and four known points with their coordinates, [7, 8].

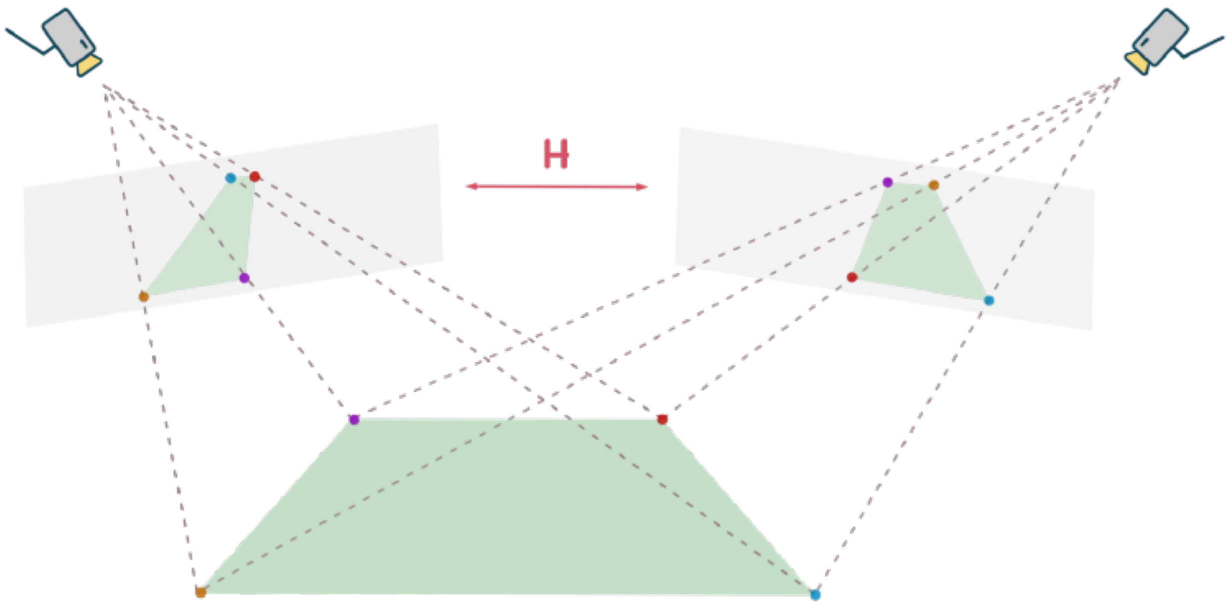


Fig 5. Example of camera perspective change using homography [R5]

A homography matrix is used to achieve this pose estimation, which is a 3x3 matrix that relates the transformation between two planes by using 8 degrees of freedom (see equation (1.3) and Fig 6). After calculating the h_{nm} coefficients, a Direct Linear Transform (DLT) algorithm is applied to the image, and the view is then converted to a desired pose, [7, 9].

$$s\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1.3}$$

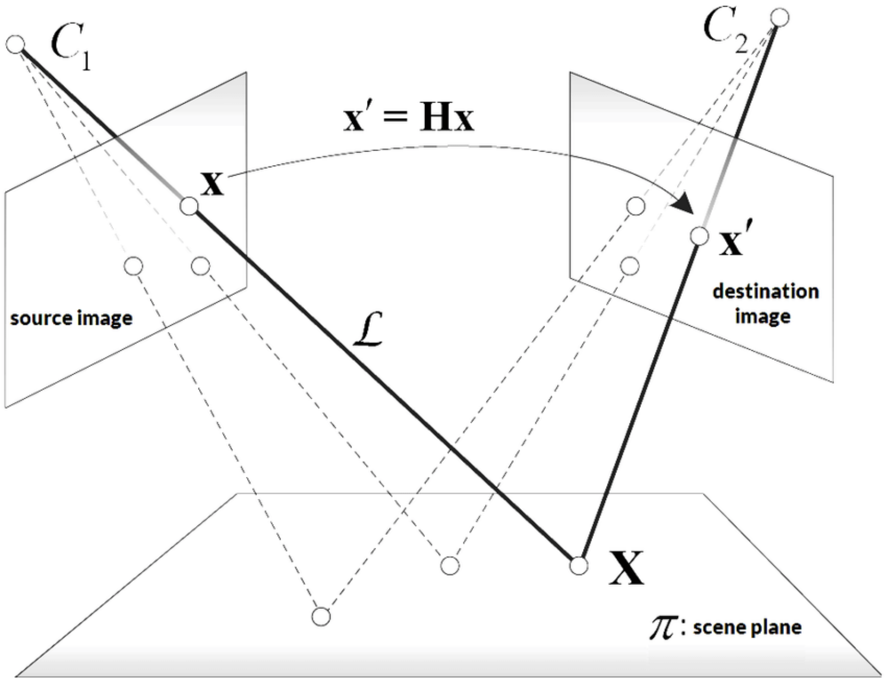


Fig 6. Planar view transformation using homography [R6]

1.4. Occupancy Grid Mapping

Occupancy grid mapping is a technique for generating environment maps from sensor or camera data. Its algorithm works by taking in the detected position of obstacles and placing its position on a grid. After gathering and scanning the environment, a grid map of the environment is captured. This technique is often used in autonomous vehicles, as it allows them to map the surrounding environment and detect any obstacles, along with their proximity. An example of the calculated occupancy grid for an autonomous vehicle is provided in Fig 7, [10].

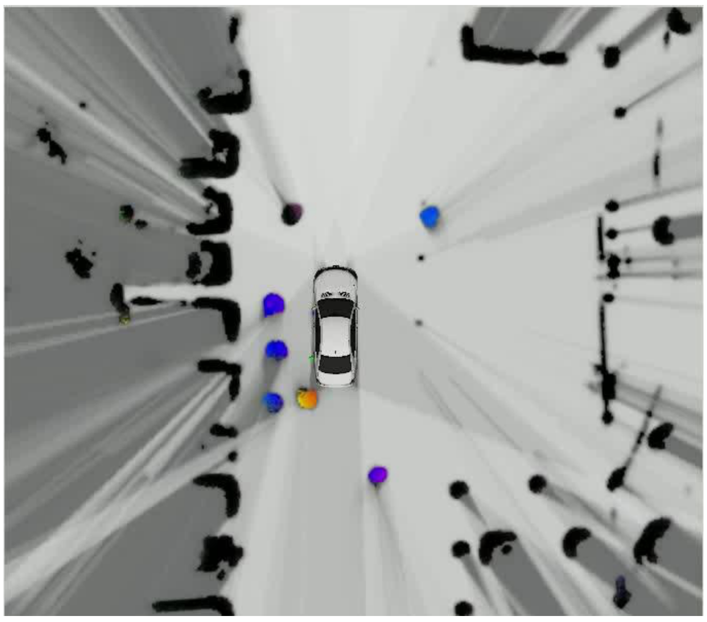
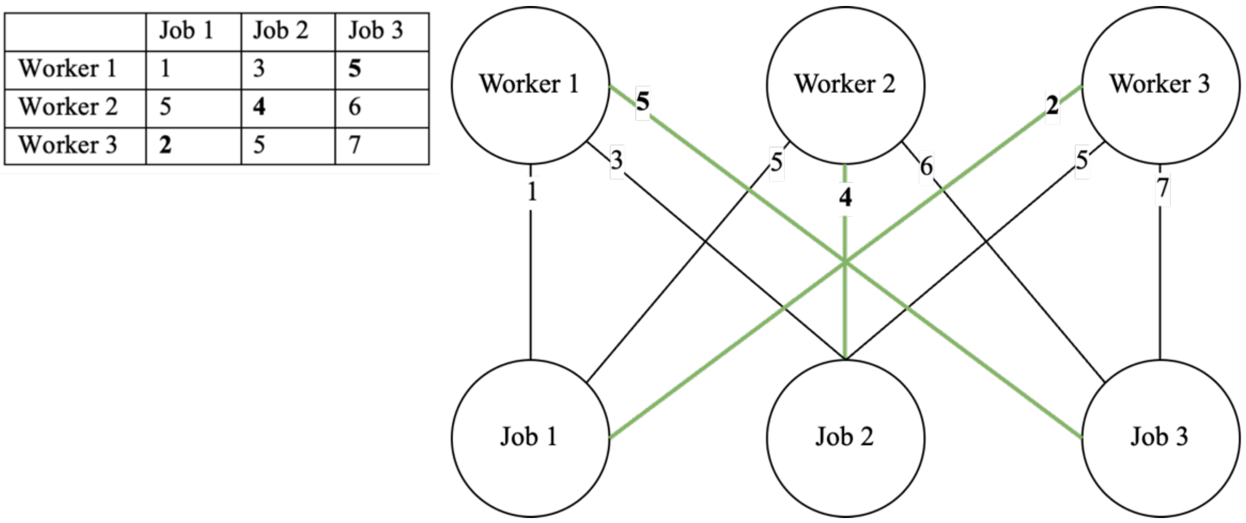


Fig 7. Occupancy grid map [R7]

1.5. Hungarian Algorithm

The Hungarian algorithm is a combinatorial optimisation technique used to figure out the most likely solutions for assignment problems. It was initially developed by Harold Kuhn in 1955, with further refinements added in the following years by James Munkres. The algorithm utilises graph theory principles to optimally pair objects from different sets in a way that affects a given cost function. An example of the assignment problem is provided in Fig 8, where the main problem is assigning workers to specific jobs to take the least amount of time, [11].



	Job 1	Job 2	Job 3
Worker 1	1	3	**5**
Worker 2	5	**4**	6
Worker 3	**2**	5	7

Fig 8. Example of an assignment problem

Furthermore, the Hungarian algorithm operates on a cost matrix foundation, where the elements reflect the cost of assigning each object in one set to another one in a different set. By identifying the lowest or highest cost of assignment between a pair, the algorithm can efficiently pair the specified objects from other datasets. Additionally, it operates on an iterative level, where the system modifies the cost matrix in each iteration, thus optimising the final solution, [11].

1.6. PyTorch Library

PyTorch library is a deep learning network/framework fully implemented in the *Python* programming language. It allows creating, training, and testing of various deep learning models, such as *Convolutional Neural Network* models. *PyTorch* is also an open-source library, thus allowing it to be used in a wide range of applications and be distributed, [12].

As it is implemented in *Python*, the creation or interaction with a neural network model is based on object-oriented programming and eases the use of the framework itself. Despite the *PyTorch* library having various pre-trained CNN models accessible to the user, it also allows the implementation of a custom neural network model sequentially or layer-by-layer, describing the model structure and architecture. It also features various optimisers and criteria that the user selects for the custom model, [12].

The fundamental workflow for working with *PyTorch* is provided in the following structure, [12]:

1. Load a dataset and process the data/images by performing various transformations (e.g., turn data into tensors).

2. Build or select a neural network model and pick a loss criterion and optimiser.
3. Create a training loop, which uses propagation to update the model weights. Hyper-parameters are also selected.
4. Run the training loop, which fits the model to the data, makes a prediction, and evaluates the result. Afterwards, adjust the weights before the next training loop.
5. Validate and test the trained model.
6. Save the trained model and its adjusted weights.

1.7. OpenCV Library

OpenCV library is a framework used for computer vision and machine learning, and it allows the application of many known algorithms to video and images. As the algorithms are all optimised for the library, the *OpenCV* library is more utilised in real-time image and video processing applications, such as object detection, recognition, view transformations, etc. An example of *OpenCV*-implemented feature matching detection is provided in Fig 9, [13].

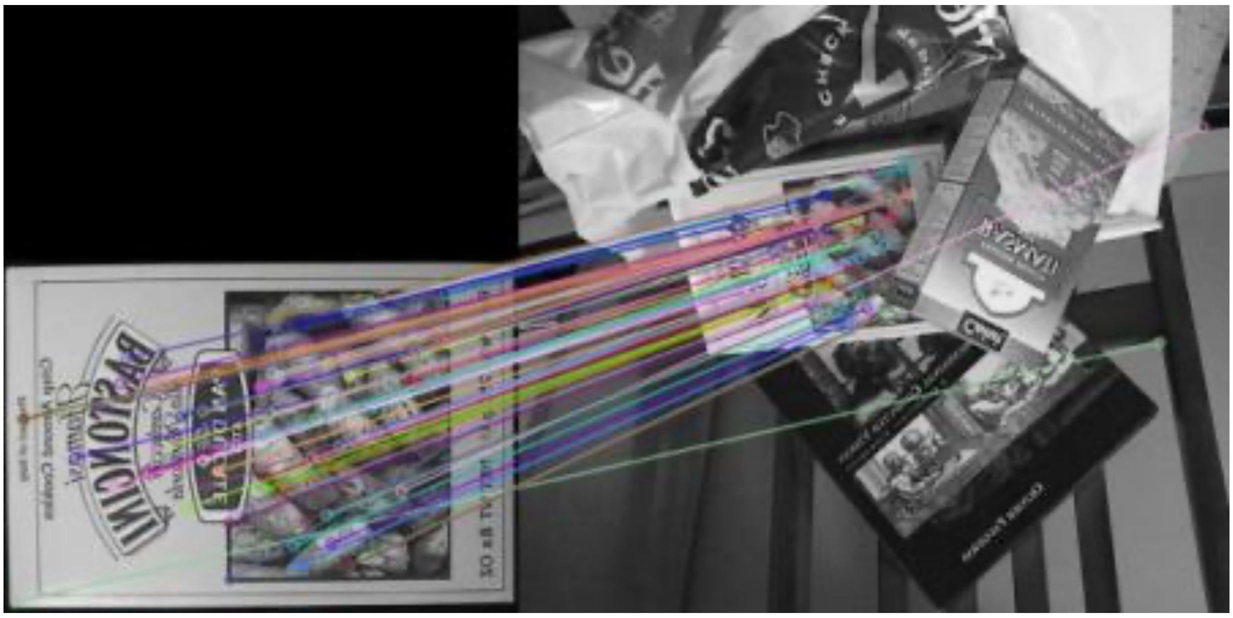


Fig 9. *OpenCV* feature matching example [R8]

The OpenCV library is implemented in the C++ programming language. However, it has bindings and interfaces for other programming languages like Java and Python. This enables the library to be versatile in many applications, as well as allows the utilisation of GPU hardware acceleration for faster image and video processing times, [13].

1.8. Artificial Neural Networks (ANN)

Artificial neural networks are computational networks whose structure is inspired by biological neural networks found in animal brains. It usually consists of an input layer, several hidden layers, and an output layer. Each layer has multiple artificial neurons, and the layers are interconnected between neurons with different weights. The neurons are specific mathematical functions and have many inputs and outputs. Each input can have different weights, which are summed and passed through an activation function. Depending on the application, these transfer functions are usually non-linear and can have other shapes. The most often used activation functions are shown in Fig 10, [14, 15].

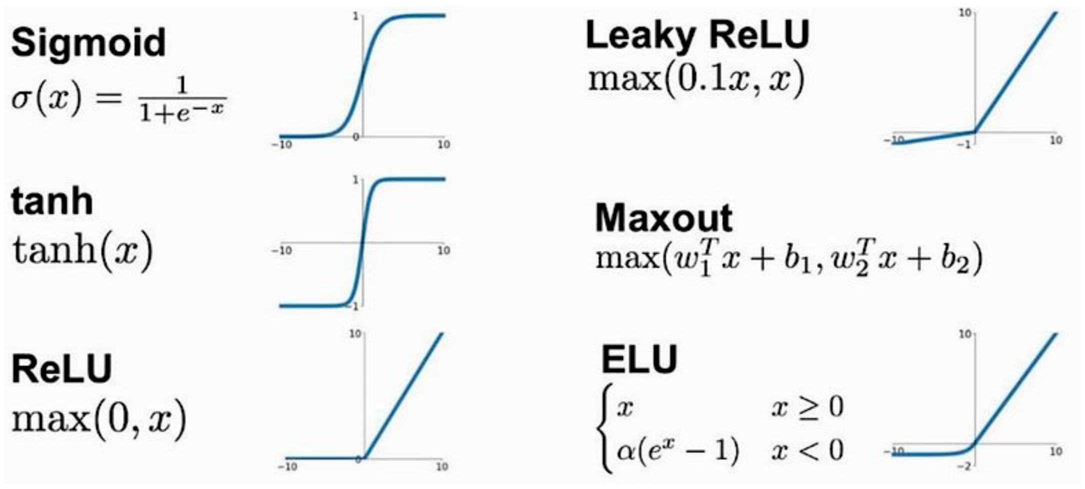


Fig 10. Popular ANN node activation functions [R9]

The training of an artificial neural network is performed by automatically adjusting each of the weights to minimise the error. To achieve this, a training dataset is provided to the network, and algorithms try to adapt the neuron learnable parameters to each element to get a more accurate result for each dataset element. The algorithm for adjusting the weights to compensate for the error is called backpropagation. This method utilises the error surface, which is a map of the loss function with respect to the weights. Backpropagation computes the derivative of it and updates the weights by following the path of gradient descent. This type of neural network is called feedforward, and its schematic is provided in Fig 11, [14, 15].

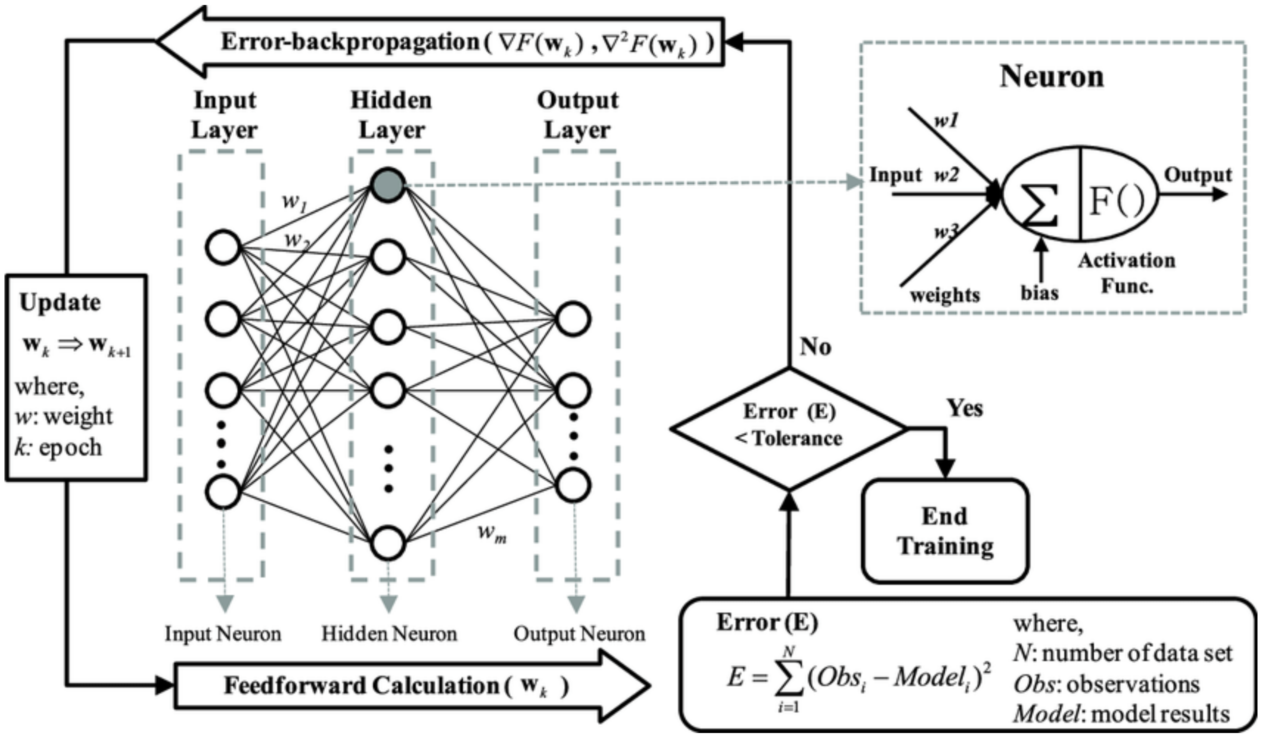


Fig 11. Schematic of a feedforward neural network with backpropagation algorithm [R10]

Multiple hyper-parameters are used to improve and optimise a neural network to a specific dataset or application, which can be adjusted depending on the desired result, [16]. Important ANN hyper-parameters are:

- *Number of hidden layers* – controls the hidden layer count. Increasing this parameter can lead to higher accuracy. However, the computational cost also increases, leading to slower model training. The increase of hidden layers can also be unnecessary, as it is possible that the model reached a plateau and will not yield better results.
- *Learning rate* – adjusts the backpropagation step size when learnable parameters are being optimised. Depending on the learning rate, the backpropagation algorithm can converge on a local minimum and not improve its accuracy anymore. On the other hand, if the learning rate is too high, the neural network can reach a suboptimal solution quickly and not explore other options.
- *Momentum* – specifies the weight of previous backpropagation steps and the directions for the current step. A high momentum value will cause previous steps to have a more significant influence over the direction of the current step and vice versa.
- *Activation function* – selects the mathematical non-linear function used in each neuron. More complex functions cause the model to train longer, whereas simple transfer functions such as ReLU will train quicker and produce adequate results.
- *Batch size* – controls the sample size of the training dataset on which the model is trained. Smaller values will produce better results; however, the training time will also increase. However, too high of a batch size can cause the model to be overgeneralised, which will cause the accuracy of new data classification to drop.
- *Epochs* – represent the number of iterations the model will train on the dataset. With each epoch, the accuracy can increase as the neuron weights are optimised more. However, a count of epochs that is too high can lead to training data overfitting, which will cause the model to be accurate on the training data but less accurate with newly received data elements. More epochs also lead to longer training time, so the number of epochs must be optimised to suit the desired performance.

All the hyperparameters are optimised to balance underfitting and overfitting and precisely specified, depending on the required training time and complexity. Fig 12 provides a general example of the prediction error's dependance on model complexity.

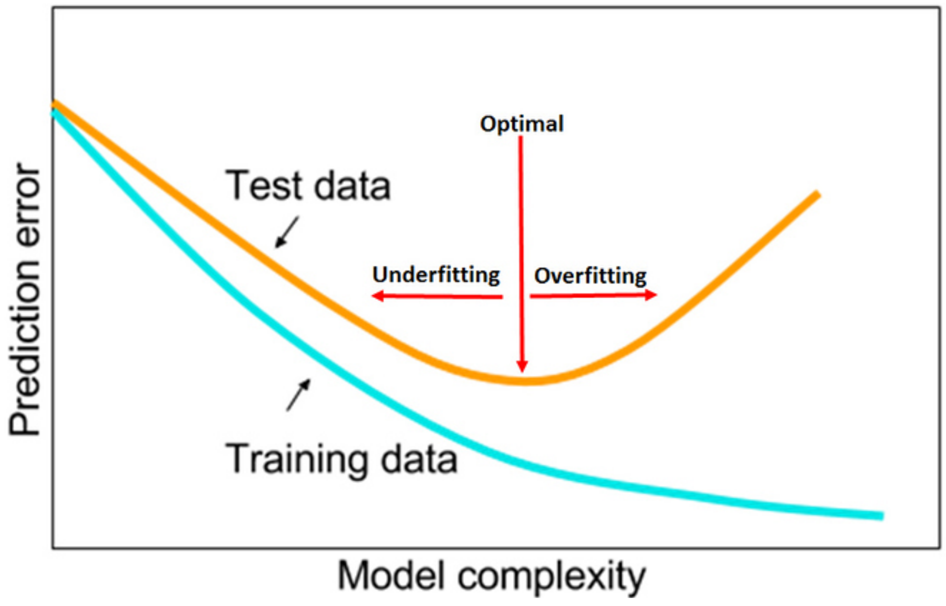


Fig 12. General example of model prediction error dependance on model complexity [R11]

1.8.1. Convolutional Neural Networks (CNN)

Apart from the feedforward neural network, other types exist, including a convolutional neural network. They were designed to deliver better results when working with datasets consisting of images, speech or audio signals. A convolutional neural network differs from conventional feedforward neural networks in that it has additional convolutional and pooling layers, [17].

The convolution layer applies a specific filter to the input image by performing a convolutional operation and passes the result to the next layer. The resulting output is a filtered and downsampled feature map, which is a more useful set of inputs to the fully connected layer. The kernel weight values are also dynamically updated and learned during the training process. However, it is important to correctly specify the hyper-parameters for the convolutional layers. The first one is the number of filters, which influence the depth of the resulting output. The next parameter is the stride, which is the step size that the filter shifts over the input matrix while convolving. The last hyper-parameter is the padding type, which specifies the type of padding used around the perimeter of the matrix, [17, 18, 19].

The pooling layer is responsible for computing the dimensionality reduction of the received matrix. It aggregates the values and populates the output matrix. There are a couple types of pooling, which differ from one another by varying the selection of element values to extract to the final output. The last layer is the fully-connected layer, responsible for the input's classification part. A general schematic of a convolutional neural network is provided in Fig 13, [17].

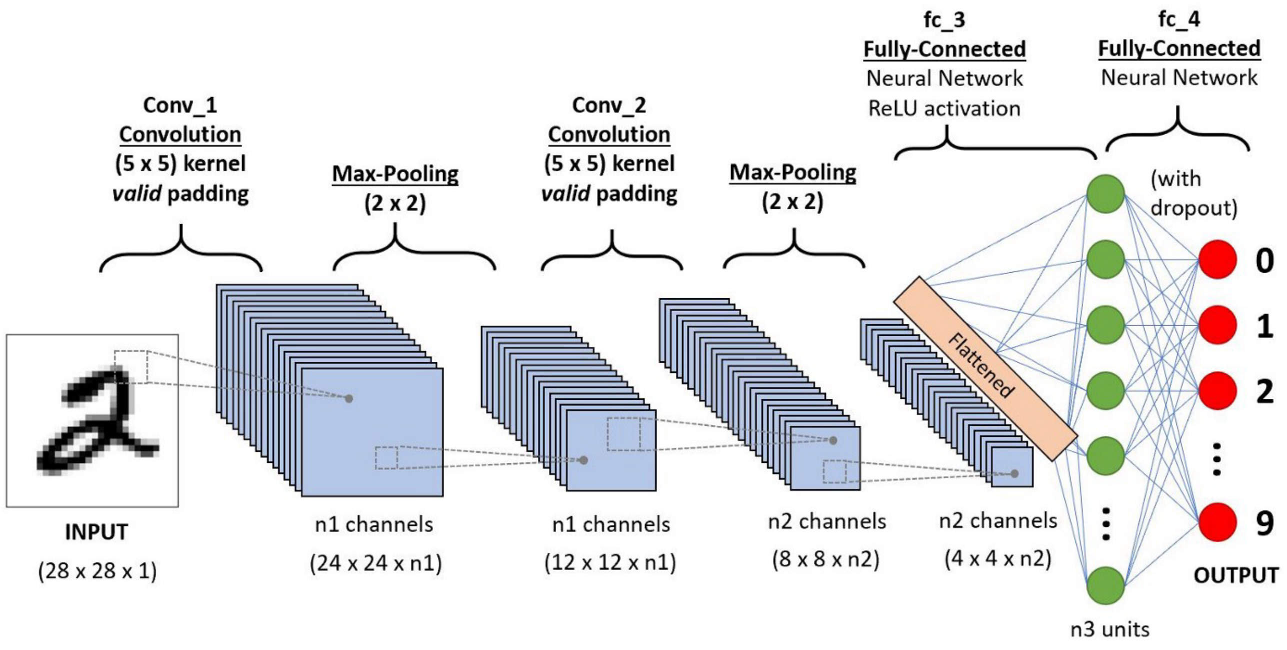


Fig 13. Example schematic diagram of a convolutional neural network [R12]

1.8.1.1. Region Based Convolutional Neural Networks (R-CNN)

The downside of a simple convolutional neural network is that it classifies the image as a whole and does not detect individual objects. To counter this problem, a region-based convolutional neural network was proposed that could lead to better object detection within a frame. The algorithm for an R-CNN consists of a region proposal, a feature extractor and a classifier, [20].

When the region proposal module receives an image, it tries to detect various regions with varied shapes by performing a selective search algorithm and extracting regions of interest (ROI). Afterwards, each region of interest is trained with a CNN network, where the most important features are extracted and transferred to a support-vector machine classifier, where the features are analysed, and a class is attached to each region of interest, [20]. The architecture of a region-based convolutional network is displayed in Fig 14.

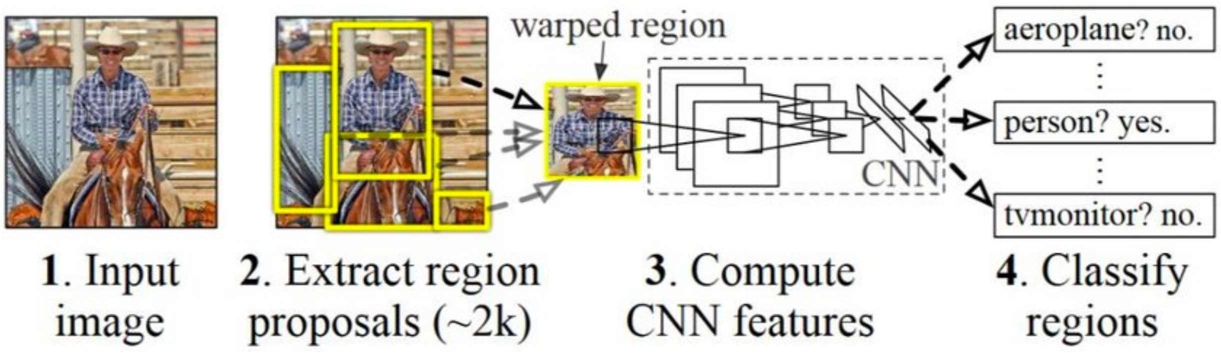


Fig 14. Region-based convolutional network architecture [R13]

However, this solution has some drawbacks, and one is that it takes a long time to train such a network, as each image would have around 2000 regions of interest per single image. Each of these regions would need to be individually trained, thus this algorithm is not very efficient. To counter these drawbacks, a Fast R-CNN model was proposed, which changed the need for feature extraction using a CNN for every single region of interest. Instead, it employs a single neural network execution for the whole image, afterwards slicing each region of interest bounding box from the CNN output and classifying the slices. This solution computes the classification task much quicker compared to the original proposition, [20, 21].

To further improve the performance of the Fast R-CNN method, a Faster R-CNN algorithm was devised, which omits the Selective Search method for finding the region of interest. To achieve this task, the ROI extraction was integrated into the neural network. This change reduces the model training and prediction time consumption even more, making the Faster R-CNN widely used in real-time object detection tasks, [22]. The architecture of this model is shown in Fig 15.

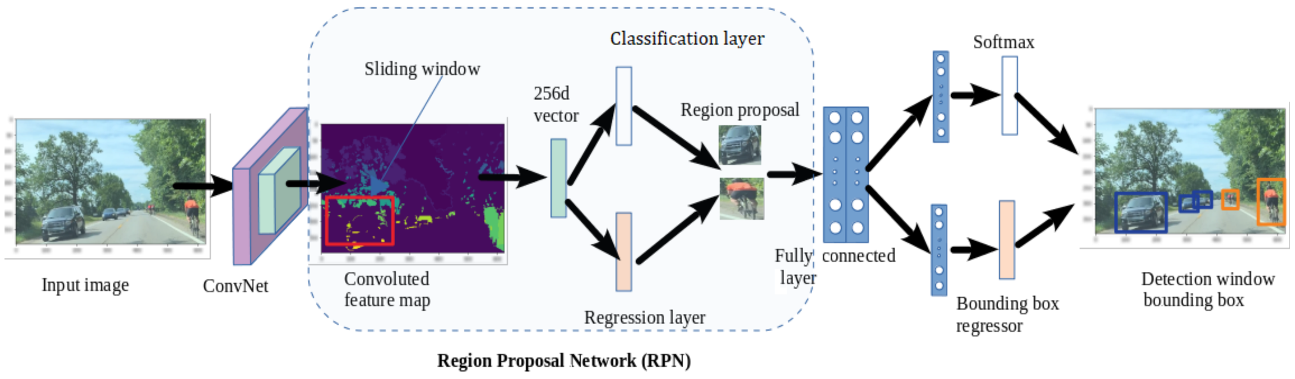


Fig 15. Schematic diagram of a Faster R-CNN model [R14]

1.9. ImageNet Dataset

ImageNet is another widely used large-scale dataset designed for image recognition research. It was created and released by data scientist Fei-Fei Li, who, together with her colleagues, presented the project in 2009 at the "Conference on Computer Vision and Pattern Recognition.". The dataset contains over 14 million images, all of which have been hand-annotated using external crowdsourcing services. The category count exceeds 20 thousand, with most containing at least a couple of hundred images for recognition, [23]. An example of the pictures contained in the ImageNet dataset and their assigned categories is provided in Fig 16.

Along with the release of the dataset, a yearly competition called the "*ImageNet* Large Scale Visual Recognition Challenge" was introduced. The contest has been integral to advancing the field of computer vision and object detection/classification, as it played a crucial part in developing models like *AlexNet*, *LeNet*, and *ResNet*, [24, 25, 26].

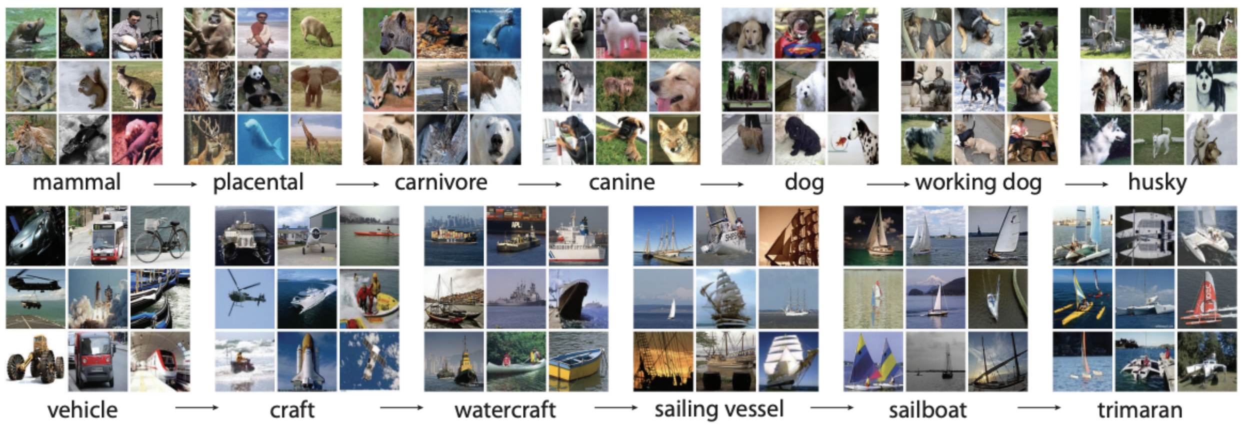


Fig 16. Example of specific *ImageNet* categorised images [R15]

The main disadvantages of the *ImageNet* dataset are its potential for bias and the possible environmental impact. The former can be caused by subjective labelling and annotation, as the hand annotations introduce a certain level of human error and bias. The root cause of the environmental impact is the massive amount of data residing in the dataset, as the model training performed on such datasets requires immense computational resources. Overall, the advantages of the extensive *ImageNet* dataset outweigh the disadvantages, as it accelerates the advancement in the machine learning field, [27, 28].

1.10. YOLO Algorithm

YOLO algorithm is a real-time object detection system created by Joseph Redmon and Ali Farhadi in 2015. The algorithm can detect objects in real-time with high accuracy. *YOLO* stands for You Only Look Once. The algorithm is based on the principle of detecting objects in images by looking at them only once, making it extremely fast and efficient. The algorithm is implemented in a neural network trained on a large dataset of images. In training, the network learns to identify objects in pictures and can then be used to detect and classify them in real-time. Because of its unique structure, the *YOLO* algorithm is one of the most accurate object detection algorithms. It can be used for various applications such as security, surveillance, and autonomous driving, as it can predict multiple bounding boxes and classes in a single image, [29, 30]. An example of a *YOLO* algorithm prediction output is provided in Fig 17.

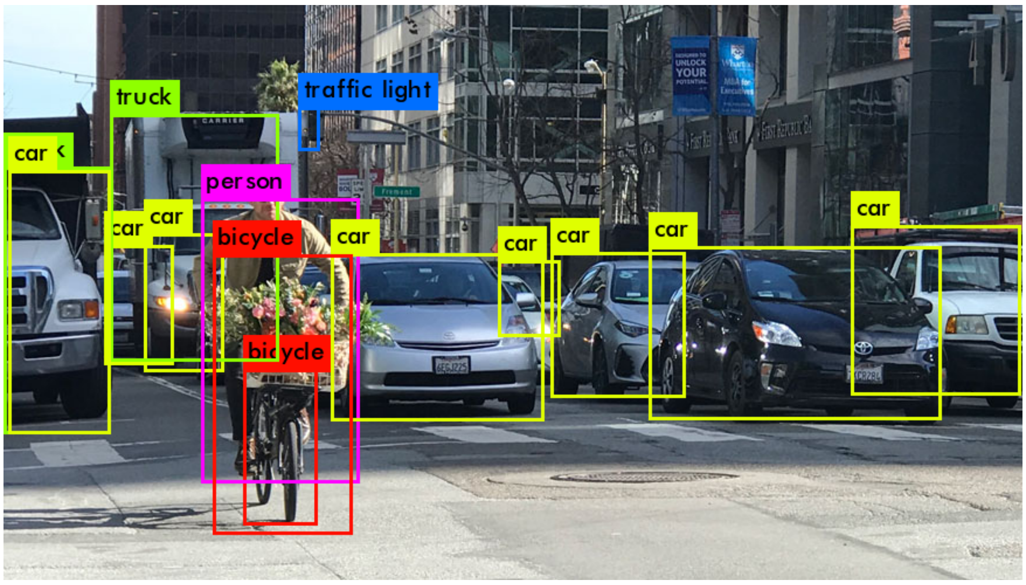


Fig 17. *YOLO* algorithm example output prediction [R16]

The *YOLO* model uses only one convolutional neural network to predict all the bounding boxes and classes, and a single forward propagation pass is enough to get a prediction output. This also speeds up the training time and, because of its structure, it also learns generalisable representations of objects, allowing the pre-trained model to be used efficiently in various applications, [29, 30].

Regarding the architecture of the *YOLO* model, it features 24 convolutional layers with two fully connected layers, and it works by first dividing the image into a grid, where each grid box produces bounding boxes together with the class confidence score. Each bounding box has five predictions: the centre coordinates of the bounding box, the width and height of the bounding box, and the confidence score, [30, 31]. The architecture of the *YOLO* model is provided in Fig 18.

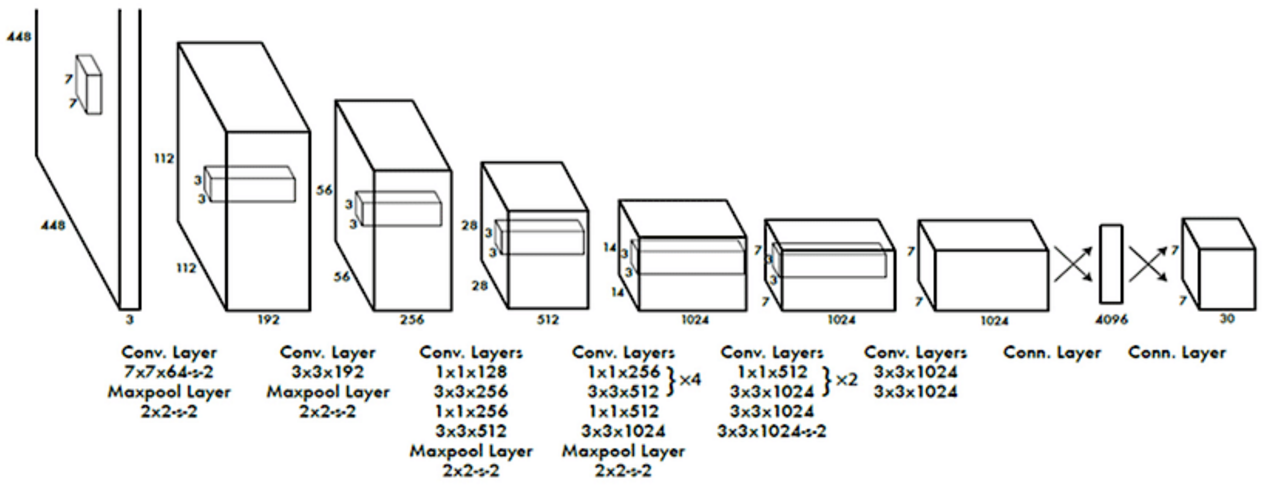


Fig 18. *YOLO* model architecture [R17]

Since the inception of the *YOLO* model, several new iterations have been created and maintained. Each iteration improved the accuracy of the model, as well as the efficiency. Most notable iterations include *YOLOv2*, *YOLOv3*, *YOLOv4*, and *YOLOv5*. The models were significant improvements over the original *YOLO* model and could even detect much smaller objects than the previous model, [30, 31].

1.11. TinyNet Model Family

The *TinyNet* model family consists of highly compact deep-learning architectures designed to be very efficient and deployable on various devices with limited computational capabilities. These models achieve their efficiency by using various optimisation techniques to reduce the model size and complexity while maintaining an adequate level of performance and accuracy, [32].

TinyNet models feature fewer parameters, making them much faster to train and run than larger models. They also feature efficiency-focused architectures and implement various innovations like neural architecture search, quantisation, and pruning. Lastly, the *TinyNet* model family is known for its ability to run on mobile devices, where processing power is often limited, [32].

The main *TinyNet* models consist of 5 distinct model versions (A, B, C, D, and E) arranged in a descending order of parameter count and accuracy. Mainly, *Model A* denotes the most accurate variant, with subsequent models reducing the parameter count and accuracy. Each model's classification accuracy on the ImageNet-1000 dataset is provided in Fig 19, [32].

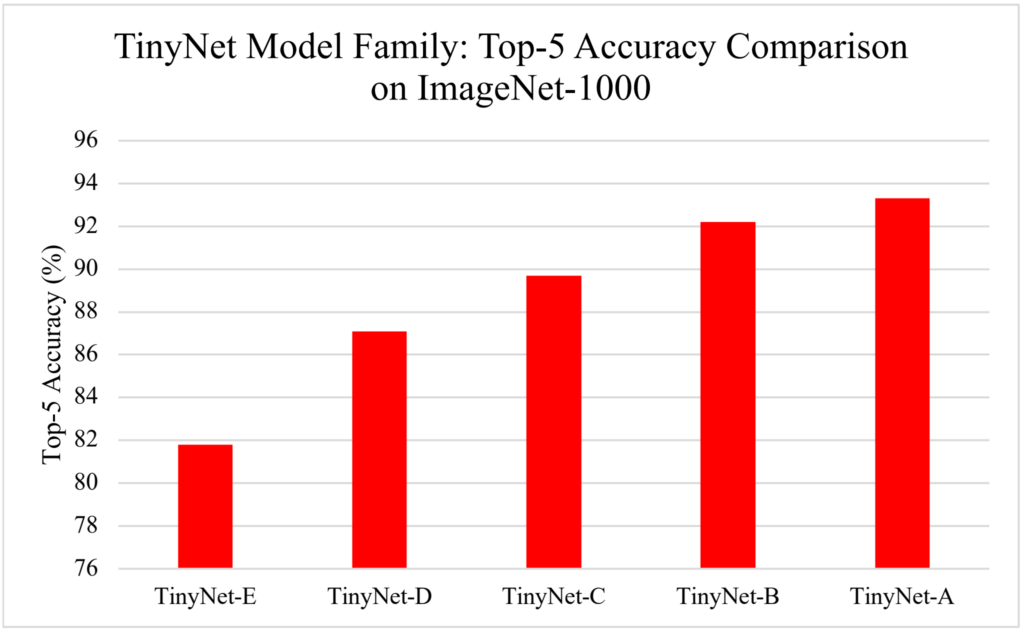


Fig 19. TinyNet Model Family: Top-5 Accuracy Comparison on ImageNet-1000

1.12. MVDet Multi-View Detector

MVDet is an algorithm designed for person detection, where multiple cameras are capturing the same area from different angles. Its multi-view approach successfully reduces the adverse effects of occlusion and varying perspectives that often negatively impact traditional single-view person detection systems. By utilising multiple cameras, *MVDet* accurately detects and localises individuals in various environments, even when they are obscured in some views, [33, 34].

The main advantage of the *MVDet* model is its feature perspective transformation (FPT) section, which aligns extracted features from different camera feeds onto a common plane. This allows the model to adequately combine information from multiple perspectives and enhance the detection accuracy. Moreover, *MVDet* utilises an anchor-free design to eliminate the need for predefined anchor boxes. This helps the model to better adapt to varying object scales and aspect ratios, [33].

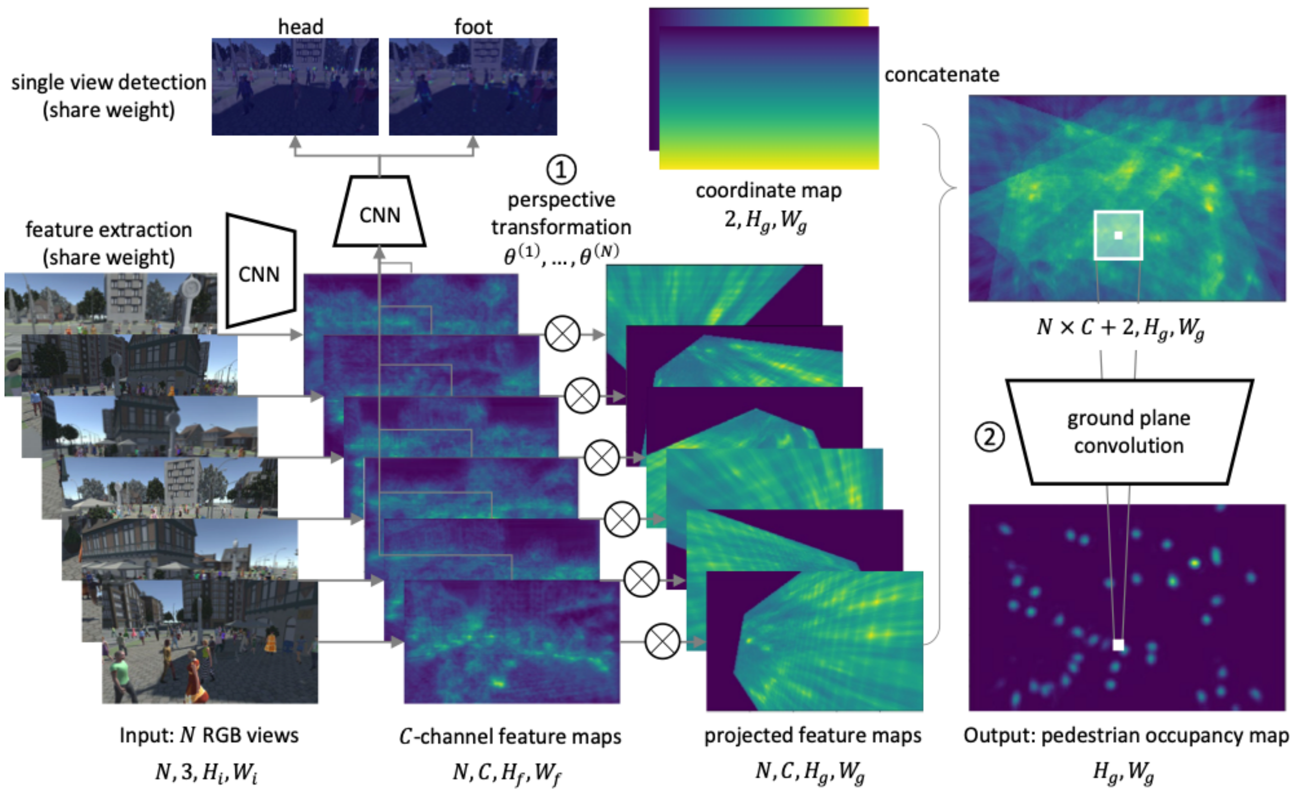

head
foot
single view detection (share weight)
CNN
feature extraction (share weight)
CNN
① perspective transformation $\theta^{(1)}, \dots, \theta^{(N)}$
coordinate map $2, H_g, W_g$
concatenate
$N \times C + 2, H_g, W_g$
② ground plane convolution
Input: N RGB views $N, 3, H_i, W_i$
C-channel feature maps N, C, H_f, W_f
projected feature maps N, C, H_g, W_g
Output: pedestrian occupancy map H_g, W_g

Fig 20. *MVDet* architecture [R18]

As shown in Fig 20, the *MVDet* architecture consists of a backbone network for feature extraction, followed by the feature perspective transformation for a final feature map fusion. Afterwards, a convolutional neural network is used to predict the person bounding boxes from the fused feature maps. The output of the system is provided in a pedestrian occupancy map, [33].

1.12.1. EarlyBird Model

*E*arlyBird is a state-of-the-art algorithm that also uses multiple camera feeds for pedestrian detection and tracking. It uses the MVDet model as its base foundation, improving its ability to track individuals across multiple cameras. EarlyBird differs from the MVDet in its feature map fusion order, as it combines them at the system's initial stage. This leads to more accurate and consistent tracking across the entire scene. The model used in the early stage as an encoder is ResNet18, from which the feature maps are extracted. Additionally, the algorithm generates a unified birds-eye view representation that provides a comprehensive perspective for object tracking. The architecture of the EarlyBird model is provided in Fig 21 and it is evaluated using the same two main datasets as MVDet: MultiviewX, a synthetic dataset featuring 6 cameras, and WILDTRACK, a real-life dataset featuring 7 cameras, [35].

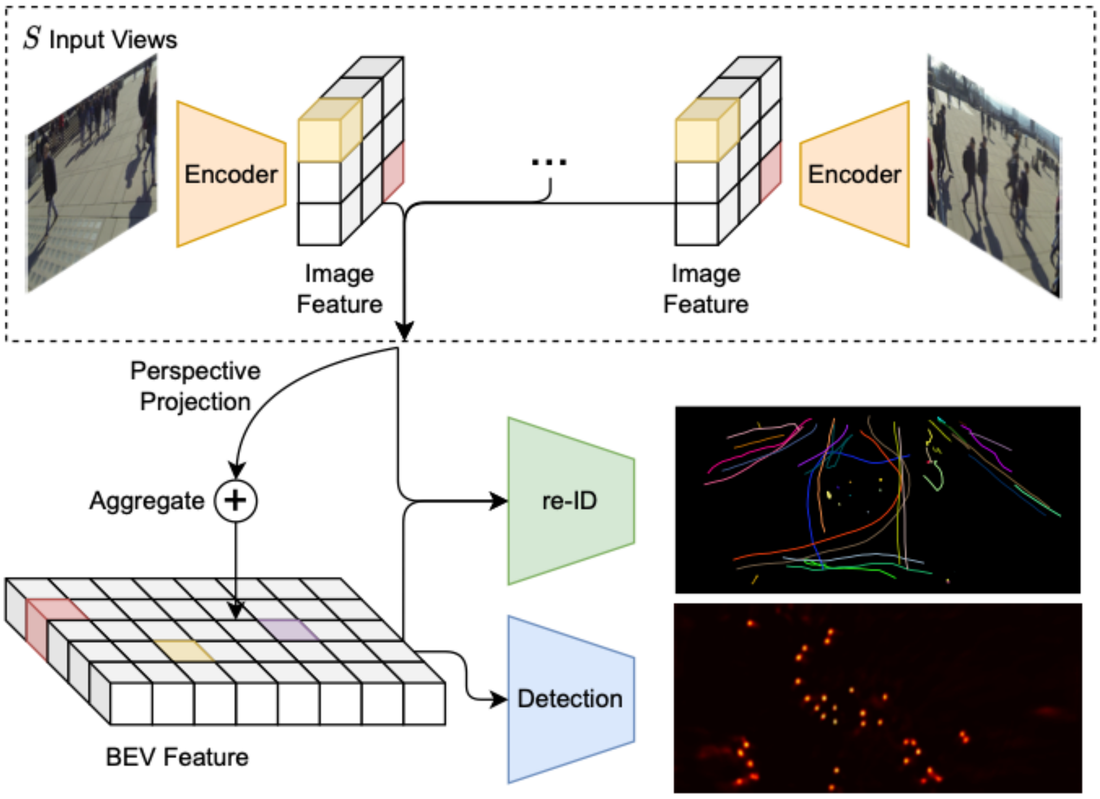


Fig 21. *EarlyBird* architecture [R19]

1.13. Methodology Summary for Multi-View Social Distancing Detection

The technical analysis of existing social distancing detection systems highlights an essential limitation in their ability to handle occlusions effectively. This reduces individual detection rate in instances where individuals partly or fully obscure another person from the main camera feed, resulting in reduced social distancing detection system accuracy.

A multi-view-based person detection system, employing two or more cameras capturing the same area from different angles, presents a solution to the occlusion issue. Instead of creating a custom algorithm using a *YOLO* model, An *MVDet*-based model (*EarlyBird*) is selected for this project to integrate multiple camera views to achieve a more accurate person detection rate in a social distancing detection system environment. Appendix 1 details additional investigation and implementation of the *YOLO* model for computer vision purposes. Additionally, as processing multiple camera feeds is a computationally intensive task, this work investigates improving the speed of detections by using a TinyNet-E model as the main backbone replacement for the *MVDet*-based model for extracting the feature maps from the provided images.

The proposed multi-view-based social distancing detection system is implemented using *Python* as the primary programming language and the deep learning library *PyTorch*. Image processing is done using the *OpenCV* library and helper packages, including *Matplotlib* and *NumPy* for data visualisation.

2. Proposed Multi-View Social Distancing Detection Solution Specifications

This chapter encapsulates the project plan for the proposed social distancing detection solution and improvements, as well as the functional/non-functional requirements, quality criteria, solution development methods and tools. Additionally, system specifications used for development and testing are provided for the proposed solution, together with the dataset analysis and testing plan.

2.1. Proposed Solution Requirements

After analysing existing solutions for a social distancing detection system, their main drawback is noted. It is bad occlusion handling, which means that if one person occludes another in the camera feed, some events may not be detected, causing the accuracy of such a system to degrade. To solve the specified issue, this project implements a solution that uses the feed of two or more cameras, capturing the same area from different angles. This should significantly reduce the inaccuracy caused by occlusion and allow for a more robust system. The main requirements for the proposed solution are that such a system should:

- Allow for multiple camera feed processing with the use of the *OpenCV* library.
- Use the pre-trained *MVDet*-based model *EarlyBird* for pedestrian detection, localisation, and tracking.
- Apply multi-layer homographic transformation to extracted feature maps for a transformed top-down view.
- Concatenate transformed feature maps and, using a set threshold, fill the occluded area of one camera with the feeds from other cameras.
- Combine the prediction maps of each prediction to reach a final prediction.
- Calculate the distance between predicted centroids and log all social distancing violations following the WHO[1] guidelines, afterwards providing a score that allows the user to compare the detection accuracy between various camera placement variants, [36].
- Output the social distancing violations with the coordinates of affected individuals in space.

As increasing the camera count leads to longer prediction and training durations, this project also investigates the possibility of replacing the first-stage backbone model with one from the *TinyNet* model family. This modification should allow the final multi-view social distancing detection system to achieve faster training and inference times, as the *TinyNet* models are less complex than the *ResNet18* used in the *EarlyBird* model.

2.2. Project Plan

The primary investigation of this project features the evaluation and comparison of a single-view social distancing detection system and the proposed novel solution for the same task. Such a system uses computer vision to detect individuals in a given space/area from provided video footage and localise their position in 3D space. Afterwards, the distance between the detected individuals is calculated, and a detection score is calculated from the detected social distancing violation events and ground truth data.

To improve upon existing solutions, the proposed solution uses an algorithm that implements multi-view video feeds for improved precision and accuracy of the final distance calculation between

[1] WHO – World Health Organisation

individuals. The use case for the proposed solution software includes the user being able to input images or videos of multiple cameras capturing the same area from different angles along with a synchronisation file that allows the system to recognise which photos belong to which frame in time. Subsequently, the user provides intrinsic and extrinsic camera calibration files for each separate camera. The intrinsic parameters describe the camera's internal characteristics, such as the principal point, focal length and distortion, while the extrinsic parameters denote the camera's position and orientation in space, [37].

Considering all required inputs were provided and valid, the system presents the pedestrian heatmap, along with social distancing violation graphs after running the predictions. Additionally, a log containing all detected social distancing violations is presented to the user, as well as the social distancing detection accuracy score and other relevant performance metrics. The use case diagram for the proposed solution is provided in Fig 22.

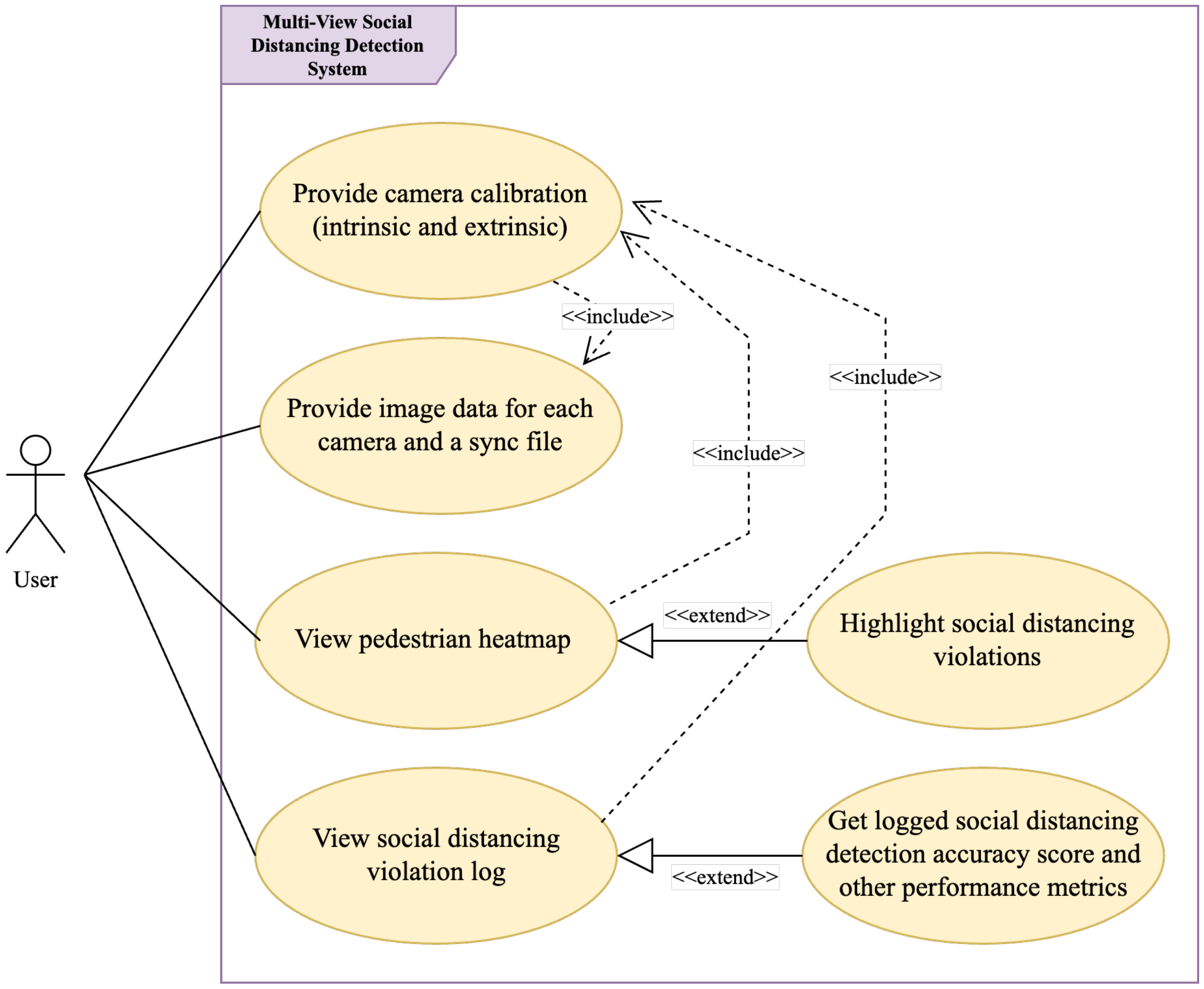

Multi-View Social Distancing Detection System
Provide camera calibration (intrinsic and extrinsic)
<<include>>
Provide image data for each camera and a sync file
<<include>>
<<include>>
User
View pedestrian heatmap
<<extend>>
Highlight social distancing violations
View social distancing violation log
<<extend>>
Get logged social distancing detection accuracy score and other performance metrics

Fig 22. Proposed multi-view social distancing detection system use case diagram

Once the use case is established and the requirements for the proposed solution are set, the inner working principles of the system are defined, which are as follows: once the user inputs the required data together with the synchronisation file, the system opens the data and saves it in the cache. Next, the user specifies the camera intrinsic and extrinsic calibration files for each camera feed. Once all required user input data is collected, the modified *EarlyBird* model extracts the feature maps from all cameras and transforms them into a top-down view for each frame. The feature maps of all cameras get concatenated and aggregated to get a final tensor that passes through a neural network decoder, after which final coordinate predictions are generated.

After receiving the predictions, the system algorithm calculates the distance between each centroid using the Euclidean distance formula. Subsequently, the ground truth points are matched to predicted points with the help of the Hungarian algorithm. After performing the necessary calculations, each social distancing violation is detected and logged if the distance between centroids is less than the set threshold. The same calculations are performed on the ground truth data as well since they are required for measuring the performance and accuracy of the detection system.

After receiving the required data, the system calculates and logs the social distancing F1 score and other relevant performance metrics of the system. As a last step, the algorithm generates a pedestrian heatmap, and violation graphs for each frame. After performing these steps, the system execution ends. The activity diagram for the proposed solution is provided in Fig 23.

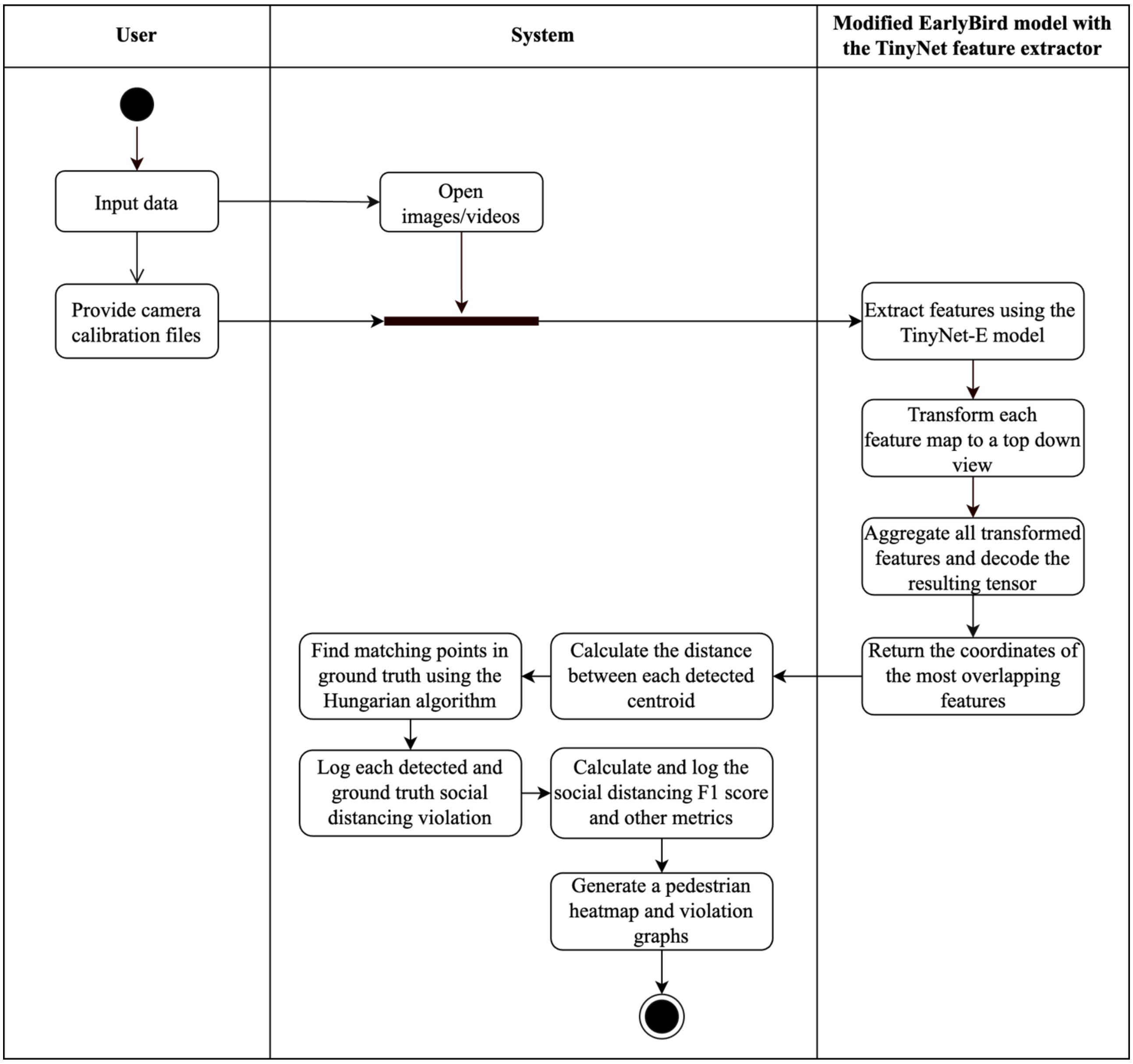

User
System
Modified EarlyBird model with the TinyNet feature extractor
Input data
Open images/videos
Provide camera calibration files
Extract features using the TinyNet-E model
Transform each feature map to a top down view
Aggregate all transformed features and decode the resulting tensor
Return the coordinates of the most overlapping features
Calculate the distance between each detected centroid
Find matching points in ground truth using the Hungarian algorithm
Log each detected and ground truth social distancing violation
Calculate and log the social distancing F1 score and other metrics
Generate a pedestrian heatmap and violation graphs

Fig 23. Proposed multi-view social distancing detection system activity diagram

2.3. Functional Requirements

To ensure that the solution and its implementation provide all necessary functionality, the following functional requirements for the solution implementation are set:

- The user is allowed to change the input images/dataset.
- The system can be trained on different datasets
- The user can set which cameras to use for the resulting output
- A heatmap indicating the pedestrian distribution of all provided scenes is provided.
- Individuals in the scene are detected and localised in space.
- A visual output of the social distancing violations is provided.
- The system can match predicted individuals to the ground truth data
- The output of multiple camera footage is combined for a final position prediction/estimation.
- The final combined top-down view window outputs a visual representation of detected individuals' locations in the scene.
- All notable predictions, estimations, and calculations are logged.

2.4. Non-Functional Requirements

To define the possible quality attributes of implemented functional requirements, constraints are placed in the form of the following non-functional requirements:

- Software implementation must work smoothly and allow for ease of use and operation.
- Formats that should be supported as the input data must include: .jpg, .png
- Each separate camera view should be provided in a separate view window with a resolution of at least 320px by 320px
- Specific camera feeds should be able to be discarded or disabled for better performance.
- Individual detection and localisation should be implemented using a pre-trained neural network.
- The combination of multiple camera footage should work by detecting common pedestrian features of one camera and combining them with the features of another camera feed.
- The final combined position prediction/estimation of each individual in the scene should be predicted at the rate of 1 frame per second or more.
- The final top-down view with combined position prediction output should be provided in a separate view window.
- The distance between each detected individual centroid should be calculated using the Euclidean distance formula.
- The pedestrian identification tracking performance should be evaluated and provided to the user.
- The calculations and results are logged after each final position and distance calculation.
- After each frame prediction and calculation update, logged values are saved to a CSV file.

2.5. Quality Criteria

The investigation of the multi-view social distancing detection algorithm includes comparisons between each method. Thus, the following multiple quality criteria for such a system/solution are selected:

- Social distancing violation detection performance: This calculates the ability to detect social distancing violation events and compares the model’s precision to detect them to the ground truth data.
- Whole-scene person detection and tracking performance: This method averages each individual's calculated detection and tracking performance in terms of reliability and predicted location deviation.
- Whole-scene prediction output speed: This gathers the time spent to process the prediction for a single scene frame.
- Training duration: This verifies the time spent to train a single model variant.

The whole scene person detection performance is evaluated with the following metrics due to them being the standard evaluation schemes for used datasets:

- Multiple object detection accuracy (MODA): focuses on the accuracy of pedestrian detection by measuring the ratio of correctly detected individuals to the ground-truth reported individuals, [38].
- Multiple object detection precision (MODP): focuses on the precision of the detection model by calculating the ratio of correctly detected individuals to the total number of predictions, [38].

Tracking performance is measured using the identification F1 score (IDF1) metric. It focuses on assessing the consistency of tracking and identifying individuals across multiple frames and considers the false positives, negatives and true positives of predicted tracked individuals. However, the IDF1 metric is susceptible to identification switches, which causes heavy penalties, [39].

To assess the training duration, all model variants are trained for 50 epochs, with the time taken to complete the training used as the final score. Regarding the whole-scene prediction output speed, it is calculated using library built-in profilers, which count the average time it takes the system to make a prediction. This metric is called model inference, [40].

The main performance metric for this project is the social distancing violation detection performance evaluation. A custom metric had to be implemented to measure the model’s ability to detect such violations. The base algorithm selected for the project is the precision, recall, and F1 scoring system, which handles the social distancing violation detection task as a classification problem, [41].

Before calculating the scores, model prediction outcomes (Fig 24) had to be defined:

- True positive (TP): Counts the presence of a social distancing violation between two predicted individuals that also matches the presence of the violation in the ground truth data.
- False positive (FP): Counts the presence of a social distancing violation between two predicted individuals when there is no such violation in the ground truth data.
- False negative (FN): Counts the absence of a social distancing violation between two predicted individuals when a violation is present in the ground truth data.

True negatives are not used when calculating the social distancing detection score, as there is no logical solution for finding true negatives in the system implementation. Furthermore, they are not needed for any precision, recall, or F1 score calculations, so this issue is not considered in this project.

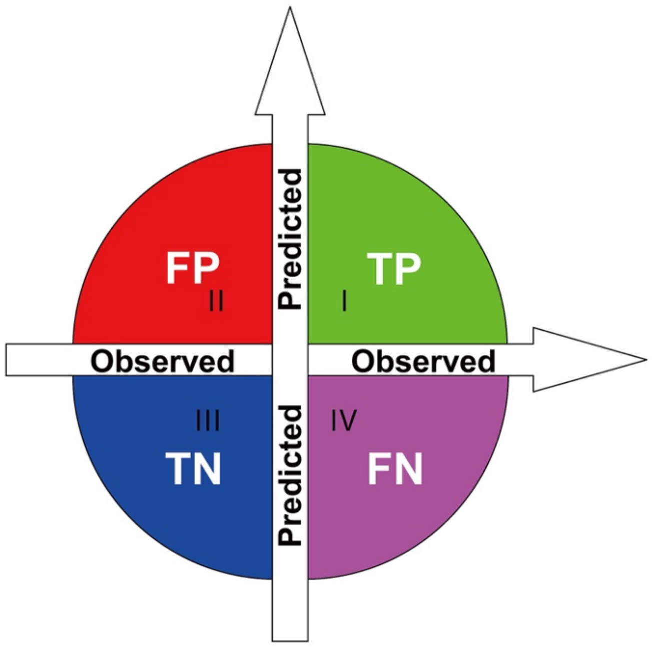


Fig 24. A confusion matrix illustrating true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) in binary classification. [R20]

Having defined the model prediction outcomes, the threshold for the distance between two individuals to count as a social distancing violation is selected as 1 metre, in accordance with the *World Health Organisation* guidelines. Lastly, the precision, recall and F1 scores are calculated using the formulas provided in the (2.1) equation. As the primary evaluation metric, the F1 score is selected as it balances precision and recall with equal weight, [36, 41].

$$Precision = \frac{TP}{TP+FP} \quad Recall = \frac{TP}{TP+FN} \quad F1 = \frac{2\times Precision\times Recall}{Precision+Recall} \tag{2.1}$$

2.6. System Development Methods and Tools

The development methods for a multi-view social distancing detection system utilise a convolutional neural network's ability to detect, localise, and segment objects. The solution also uses *CUDA* hardware acceleration for faster training and prediction times. Furthermore, all proposed backbone models use pre-trained weights to achieve higher accuracy and shorter training times. The models are acquired using the *TIMM* library, which features many widely known object detection algorithms and provides a simplified interface for their custom integration into projects, [42].

The solution achieves the top-down camera view by transforming the input image with a perspective transform operation and using the provided intrinsic and extrinsic camera calibration files supported by *OpenCV*. Regarding the distance calculation between centroid distances, a distance formula is used, provided in equation (2.2), where x_0, y_0, z_0 are the coordinates of 1st centroid and x_1, y_1, z_1 are the coordinates of 2nd centroid, [43].

$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2} \tag{2.2}$$

For the development of the proposed solution implementation, the following software tools are used:

- *Python*: It is used as the main programming language and framework for the development process. It was chosen because of its simplicity, wide accessibility, and large number of available machine-learning libraries supported by the community. Other advantages include the ease of understanding and versatility of its code, [44].
- *PyCharm IDE*: This *Python*-oriented integrated development environment is used to write the implementation code, run experiments, and compare different implementation variants.
- *OpenCV* (*Python* library): This library is used to process input and output data, which are images/videos of crowd interactions. It allows the use of a variety of algorithms for image processing, transformation, etc. *OpenCV* is also recommended as the main video processing library for real-time video footage, [13].
- *PyTorch* (*Python* library): This library is used to create custom neural networks or utilise pre-trained ones. In the development of the solution, this library is mainly used for the prediction functionality of a pre-trained object detection and localisation CNN model. As the library is implemented natively in *Python*, it is the most suitable machine-learning library for the proposed solution, [12].
- *TIMM* (*Python* library): This library contains a collection of state-of-the-art models focused primarily on computer vision tasks. All its models are created using the *PyTorch* library, thus ensuring a stable and accurate platform for model usage, [42].
- *EarlyBird* (Pedestrian localisation model): This model is used as the central task solver in the proposed system implementation. The task is to combine multiple camera views, detect each person in an area, and output the prediction in a compatible format for other libraries, which could process its outputs and perform further calculations. Its main advantage over other available multi-view models is its prediction output speed and accuracy, thus complying with the set requirements, which state that the solution output should process at least a frame per second, [35].
- *NumPy* (*Python* library): This library is widely used for performing numerical and scientific calculations in *Python*, as it allows more efficient array and matrix operations. This project utilises its capabilities when manipulating compatible data structures and performing various analysis calculations, [45].
- *Matplotlib* (*Python* library): This library is applied when there is a need for static or interactive visualisations, such as graphs and diagrams. This project uses it to create various heatmaps and provide visual data of the predicted social distancing violations, [46].
- *TensorBoard* (Visualisation toolkit): This toolkit is integrated into the project to visualise and monitor the implemented systems' machine learning metrics, such as training loss, accuracy, and evaluation metrics like MODA, MODP, etc, [47].

The system must be retrained for each camera configuration before testing can commence. This causes issues with hard drive space, as the training produces a lot of checkpoint data among the logged metrics. Due to this issue, *Google Drive* is used for storing the training data and all executed evaluation logs. This is also beneficial because the experimental part is executed on *Google Colab*, which has built-in integration with *Google Drive* and allows cloud storage to be mounted as a dedicated disk in the virtual machine, [48].

2.7. Test Environment Specifications

As a social distancing detection system that utilises multiple camera feeds requires significant computational resources for efficient performance, the test environment system specifications selected are set to a high standard. The specifications include a computer resource with these components:

- Intel Xeon Multi-core CPU
- 32 GB RAM
- NVIDIA T4 GPU with 16GB GPU memory

The main software and libraries used in the implementation and testing stage are:

- *Python* 3.11
- *PyTorch* library (latest)
- *OpenCV* library (latest)
- *TIMM* library (latest)
- *Matplotlib* library (latest)
- *EarlyBird* library (latest)
- *TensorBoard* toolkit (latest)

Additionally, helper libraries like *numpy* and *matplotlib* are used for faster calculations and graph visualisations. Furthermore, the selected system specifications ensured the solution development and testing/experimentation process ran smoothly. Nevertheless, due to the high system requirements, outsourced computing resources are utilised; the solution implementation uses the *Google Colab* platform as the primary testing and experimenting platform, as it offers powerful hardware resources for a small fee, [48].

2.8. Dataset Analysis

To successfully evaluate the implemented solution and compare it against existing solutions, two datasets will be utilised for experiments and testing. These datasets are intended for the use of evaluating multi-view crowd detection and count algorithms. However, this final project uses these datasets to calculate and detect social distancing violations and test various camera placement variants to evaluate the impact of their position on the overall model detection accuracy and inference latency.

2.8.1. "WILDTRACK" dataset analysis

The first dataset used to evaluate and compare the proposed solution against existing solutions is called *WILDTRACK*. It was created by the "Swiss Federal Institute of Technology" and meets the solution testing criteria of having multiple cameras capturing the same area from multiple angles. The dataset provides video data of an outside area with various crowds of people walking and interacting with each other. Moreover, as the static cameras are mounted low, the resulting overlap of individuals provides an additional challenge for single-view social distancing detection systems, thus allowing for more apparent algorithm evaluation, [49]. An example of the datasets captured single image is provided in Fig 25.

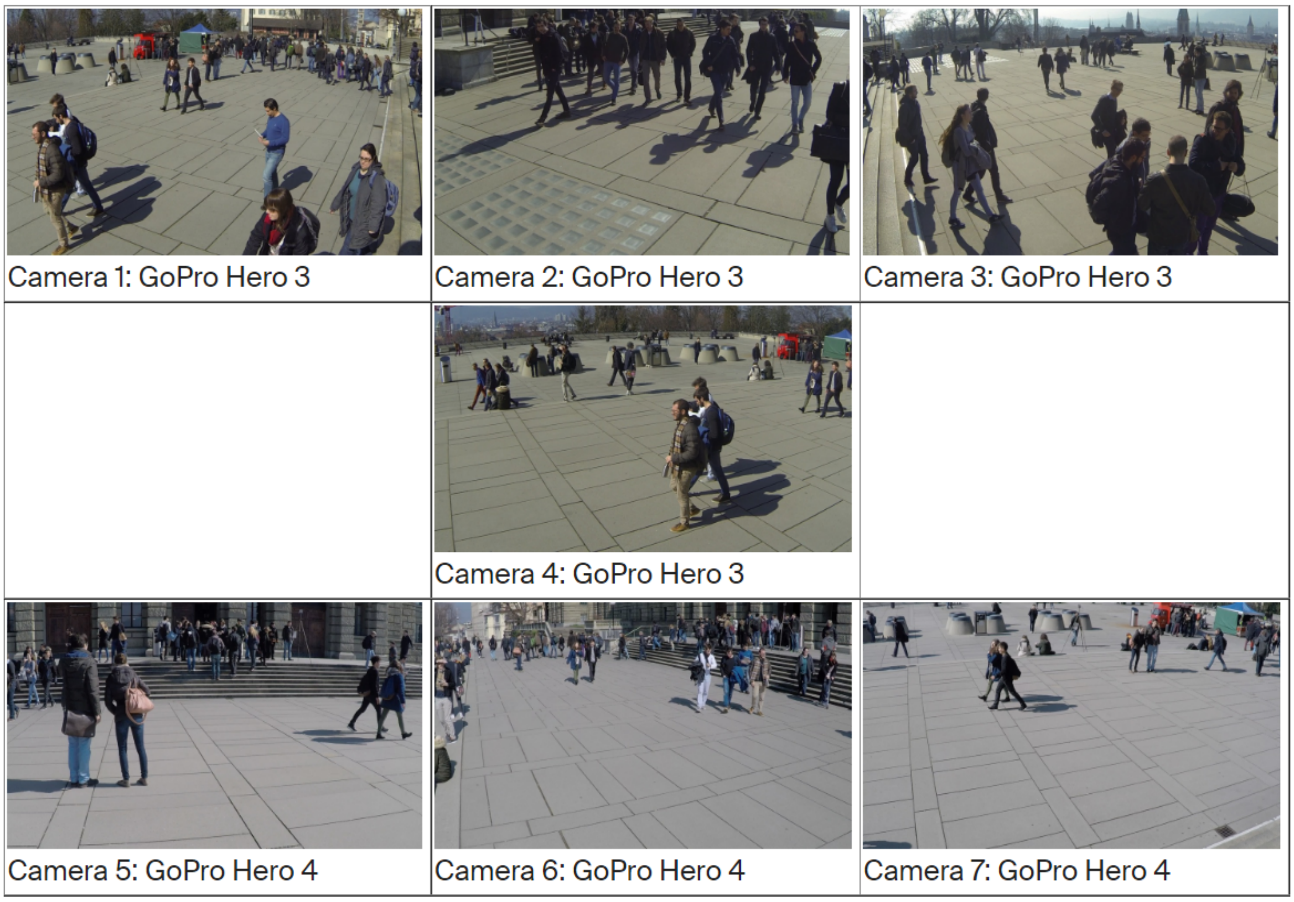


Fig 25. Example of a single data point from the *WILDTRACK* dataset [R21]

The dataset consists of the footage captured by 7 *GoPro Hero* cameras mounted in different locations and angles, along with camera calibration data and the synchronisation between the view sequences. The provided dataset footage features a resolution of 1920x1080 pixels and has extracted synchronised frames with a framerate of 10 frames per second. The calibration files for each camera also provide compatibility with the *OpenCV* library used in the proposed social distancing detection system and its projection functions, [49].

Its working area is 12 x 36m^2, and the ground truth annotations are provided for every two frames per second, which feature coordinates in space for each person in view in the captured area for each camera. Additional interpolation may be used to enlarge the annotations file, [49].

2.8.2. “MultiViewX” dataset

The second dataset that will be used for evaluation is the *MultiViewX* dataset, a synthetic dataset created with the *Unity* game engine. It features more people in the frame (40 people) compared to the *WILDTRACK* dataset and a smaller ground plane of 16 x 25m^2. The dataset provides an open area with simulated people interactions, such as walking or standing. As the number of individuals in a frame is higher, the occlusion is, in turn, also more apparent, thus increasing the need for a multi-view prediction output for higher accuracy, [33]. An example of a single data point of the dataset is provided in Fig 26.

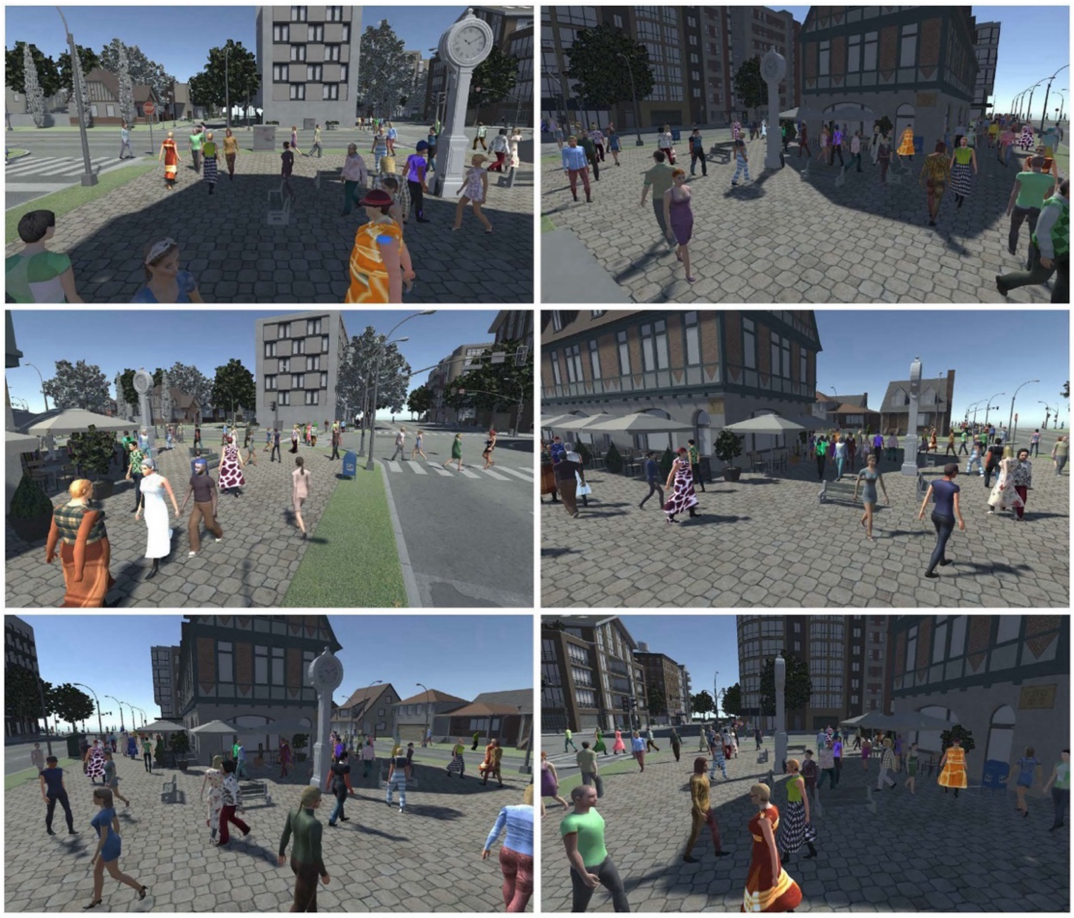


Fig 26. Example of a single data point from the *MultiViewX* dataset [R22]

As the dataset was synthetically created, its ground truth data is very accurate and provides a better insight into the evaluation of the solutions' performance. Furthermore, the dataset was created to have the same annotation structure as the *WILDTRACK* dataset, and it has the same 1920x1080 resolution and a synchronised annotation framerate of 2 frames per second. These similarities to the *WILDTRACK* dataset provide easier compatibility and improved testing/evaluation results of the existing and proposed solutions, as the evaluation and testing techniques can be used identically for both datasets, [33, 49].

2.9. Testing Plan

The project testing plan consists of multiple parts, including the proposed solution implementation development testing. It consists of continuous code testing for each iteration and revision of the code, using unit and integration tests. These tests ensure that each code revision keeps the structure and working principle of the software itself intact. As integration tests are used, code testing begins after the first working iteration of the solution.

The second part of the testing plan concerns the proposed solution's performance and its testing against other solutions. The testing of the implemented multi-view social distancing detection system involves using the set quality criteria for assessing and testing the performance of the implemented solution. Furthermore, evaluation is performed with the use of 2 separate datasets, as well as by using different camera placements for the final prediction output evaluation.

3. Evaluation of Multi-View Detection Application for Social Distancing Monitoring

This chapter details the implementation of a multi-view social distancing detection system and assesses its performance using specified metrics. The results of this evaluation are presented in tables and graph visualizations.

3.1. Solution and Improvements Implementation Details

This subchapter describes implementing and refining the proposed multi-view social distancing detection system by incorporating modifications to the *EarlyBird* detection module, integrating additional backbone models, developing a custom social distancing metric, and developing performance evaluation tools. The implementation of these modifications allows this project to successfully assess the viability of using a multi-view based detector in social distancing monitoring systems.

3.1.1. EarlyBird Detection Module Adaption

The *EarlyBird* pedestrian detection module, designed for multi-view object detection, is selected as the base for the proposed system. However, the original module lacks flexibility in choosing different camera configurations. To address this issue, modifications are made to enable the selection of specific cameras for both training and evaluation.

Initially, the base *EarlyBird* model used all available cameras in the dataset. However, the new modification involves modifying various source code files to replace the fixed *num_cam* parameter (e.g., *num_cam=6*) with a list of camera indices (*num_cam=[0, 1, 2, 3, 4, 5]*), as this allows for more granular control over the camera setup used during experimentation. However, due to its design, the *EarlyBird* model does not work when selecting a single camera for evaluation. To avoid this, a workaround is implemented using duplicate camera indices (e.g., *[1, 1]*) to simulate single-camera evaluation. This solution is acceptable when evaluating the system's accuracy, yet it is impossible to assess the timing metrics accurately.

3.1.2. Custom Backbone Encoders Integration

The initial stage of the *EarlyBird* model depends on the feature extraction backbone. To evaluate the impact of different feature extraction algorithms, various backbone models are integrated into the system as encoders. *EarlyBird's* default encoder is the traditional *ResNet18* convolutional neural network, so models ranging from less complex to more complex are implemented as optional encoders with different layers used:

- *TinyNet-E*: 4 layers
- *Swin Transformer*: 3 layers
- *TinyViT*: 3 layers

Different upsampling and concatenation kernels are used for all the custom encoders, as each has differently sized tensor inputs and outputs. To switch between these encoders during training and testing, there is a dedicated variable in the configuration: *encoder_name*, which declares the encoder type.

The classification heads of all encoders are also removed because they are unnecessary for feature extraction and only add to the resource overhead. This change simplifies the model and directs its computational resources towards generating detailed feature maps for the detection process.

3.1.3. Social Distancing Detection Metric Implementation

A custom social distancing metric is developed to assess the proposed solution‘s performance. This metric operates by analysing the prediction log, containing predicted individual locations for each frame, and comparing it to the ground truth data. The Hungarian algorithm is used to match predicted and ground truth pedestrian coordinates. Afterwards, a confusion matrix is calculated for each frame, which gathers the true positives (TP), false positives (FP), and false negatives (FN). Lastly, after evaluating all frames for the predicted and ground-truth social distancing violations, precision, recall, and the computed F1 score and reported alongside other metrics in the *tfevents* file format. The algorithm for calculating the social distancing detection scores is provided in Fig 27.

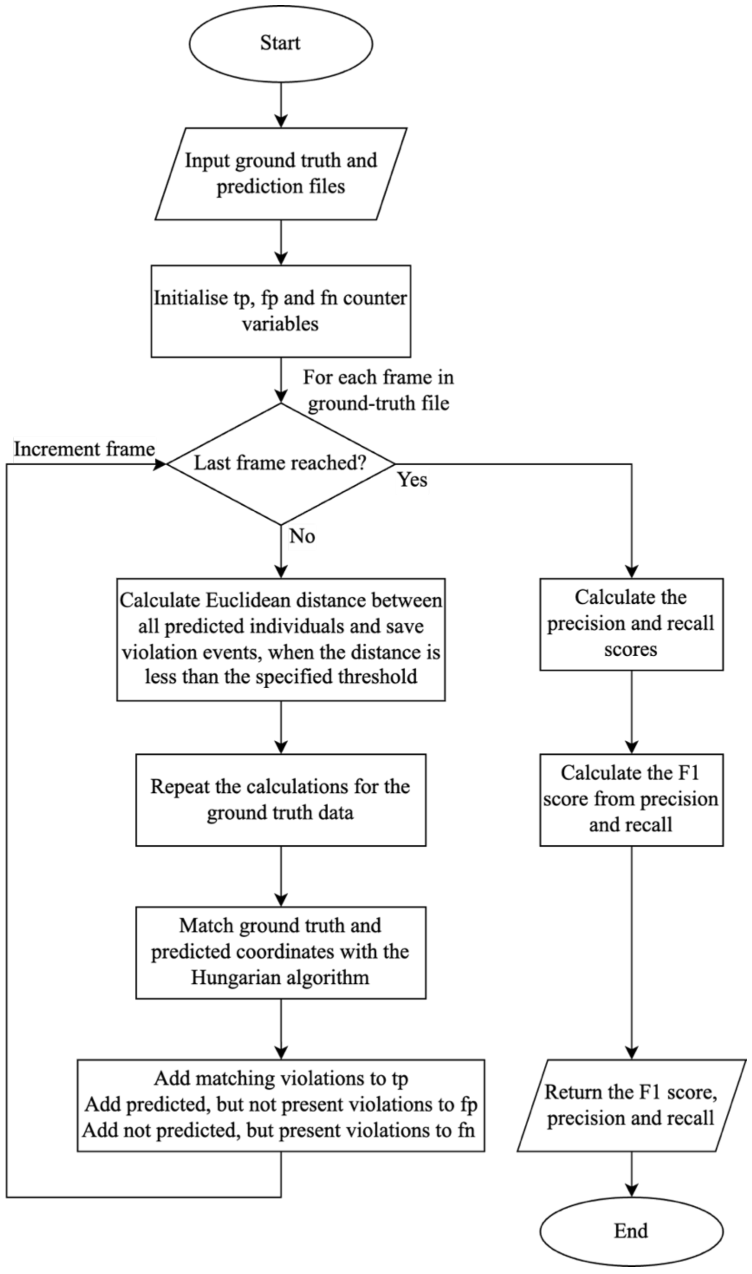

Start
Input ground truth and prediction files
Initialise tp, fp and fn counter variables
For each frame in ground-truth file
Last frame reached?
Increment frame
Yes
No
Calculate Euclidean distance between all predicted individuals and save violation events, when the distance is less than the specified threshold
Repeat the calculations for the ground truth data
Match ground truth and predicted coordinates with the Hungarian algorithm
Add matching violations to tp
Add predicted, but not present violations to fp
Add not predicted, but present violations to fn
Calculate the precision and recall scores
Calculate the F1 score from precision and recall
Return the F1 score, precision and recall
End

Fig 27. Social distancing violation detection metric calculation algorithm

3.1.4. Performance Evaluation Additions

Solutions for tracking inference and training time are implemented to assess the efficiency of the multi-view detection system. Training time is automatically recorded in the *tfevents* log file generated by *PyTorch*, while inference time is captured using the *PyTorch* built-in profiler. The profiler also monitors *CUDA* memory usage and calculates the average model inference time across all frames.

Furthermore, a custom parser is created to convert *tfevents* files into a compatible CSV format for streamlined analysis. It considers all evaluation model runs in a directory and outputs a single CSV file containing data for each investigated camera configuration. The parser algorithm, which utilises the *EventAccumulator* class for processing tfevents files, is provided in Fig 28. Moreover, to comply with the set functional requirements, *Matplotlib* is employed to generate heatmaps and visualise social distancing violations.

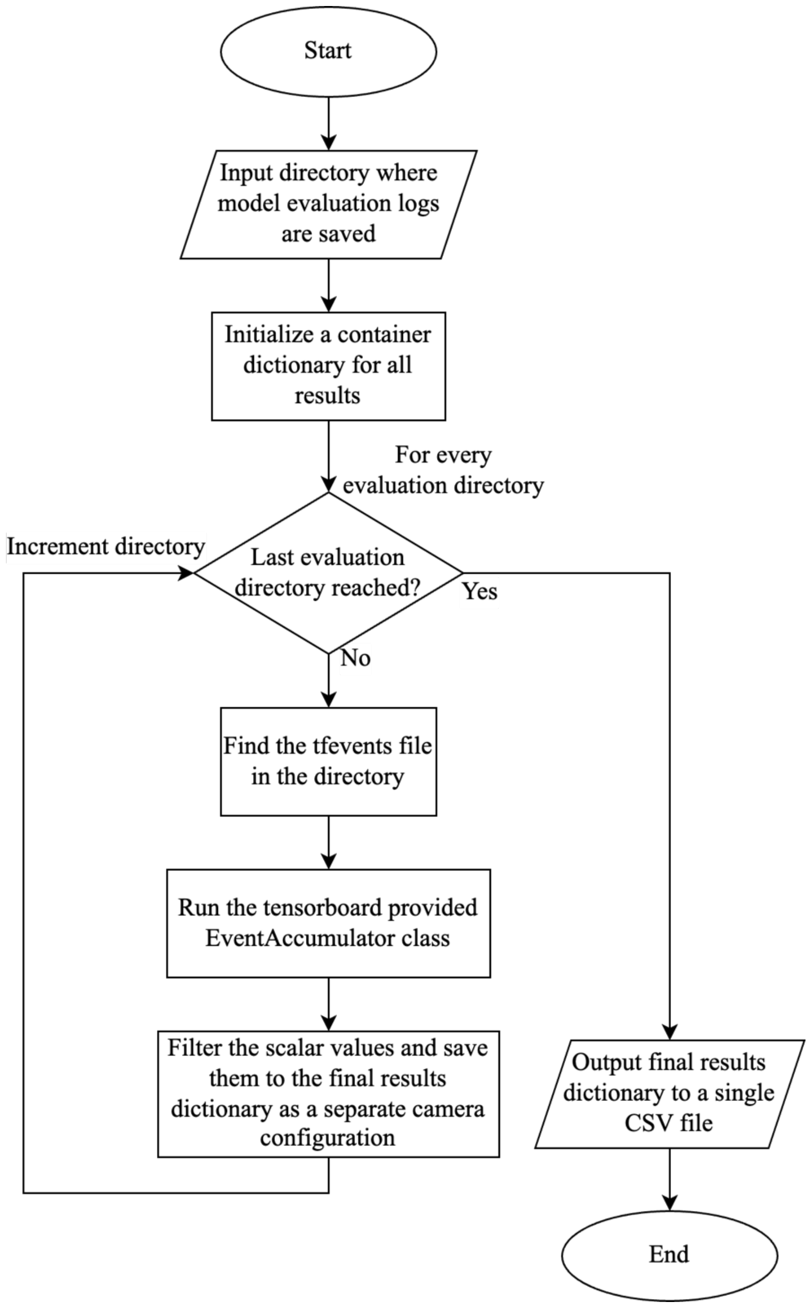

Start
Input directory where model evaluation logs are saved
Initialize a container dictionary for all results
For every evaluation directory
Last evaluation directory reached?
Increment directory
Yes
No
Find the tfevents file in the directory
Run the tensorboard provided EventAccumulator class
Filter the scalar values and save them to the final results dictionary as a separate camera configuration
Output final results dictionary to a single CSV file
End

Fig 28. Parser algorithm for converting *tfevents* files to a CSV file

3.1.5. Training and Testing Data Variants

Multiple training data variants are created to evaluate the proposed system's performance under various conditions. These variants involve varying the number and placement configuration of cameras used for predictions. Additionally, two distinct camera placement combinations are chosen for both dataset configurations to capture the area from different perspectives. The selected configurations are provided in Table 1, where camera numbers represent their location, as shown in Fig 29 and Fig 30.

Table 1. Camera placement configurations for system evaluation

Camera count	**MultiviewX**		**WILDTRACK**	
	Configuration A	**Configuration B**	**Configuration A**	**Configuration B**
1	3	2	3	1
2	1, 2	5, 6	3, 6	1, 2
3	1, 2, 3	1, 5, 6	1, 2, 3	2, 3, 6
4	1, 4, 5, 6	2, 3, 5, 6	4, 5, 6, 7	1, 2, 3, 6
5	1, 2, 3, 5, 6	1, 2, 3, 4, 6	1, 3, 4, 5, 7	1, 2, 3, 5, 6
6	1, 2, 3, 4, 5, 6		1, 2, 3, 4, 5, 6	1, 2, 3, 4, 5, 7
7	-	-	1, 2, 3, 4, 5, 6, 7	

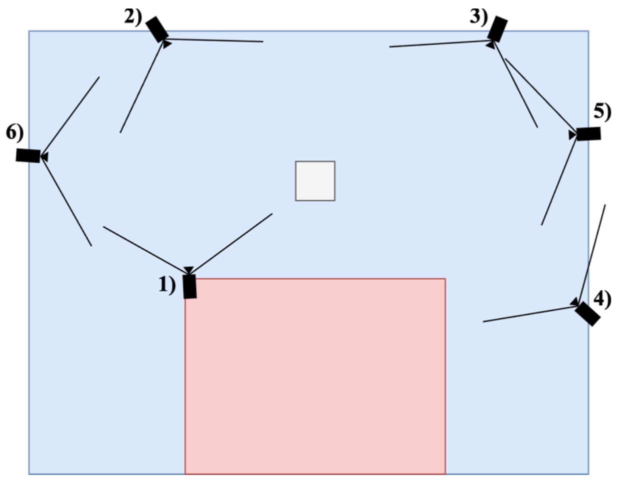


Fig 29. *MultiviewX* dataset camera positions (top-down)

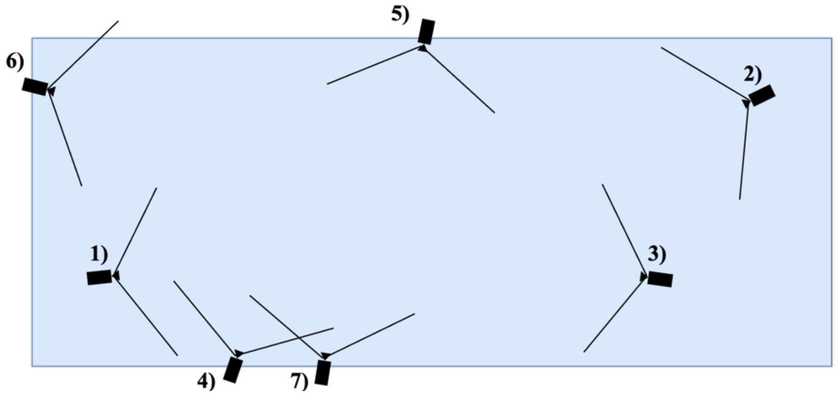


Fig 30. *WILDTRACK* dataset camera positions (top-down)

3.2. Datasets Investigation

Before the evaluations of the proposed social distancing detection system can commence, a dataset analysis is done. It begins with the verification of the dataset camera calibration files, where a visual inspection is accomplished. First, a random selection of images for each dataset is selected and loaded into system memory with the help of the *OpenCV* library. Furthermore, the calibration files are also provided to the library, and a separate point grid is created. Lastly, the point grid gets projected onto the images, using homography principles, and the final result is reviewed.

After reviewing the results of each dataset's camera calibration files, the final verdict is favourable, as using each camera's calibration file for the transformation matrix resulted in a correctly mapped point grid. Fig 31 provides an example of the verification result, while more samples can be found in Appendix 2.

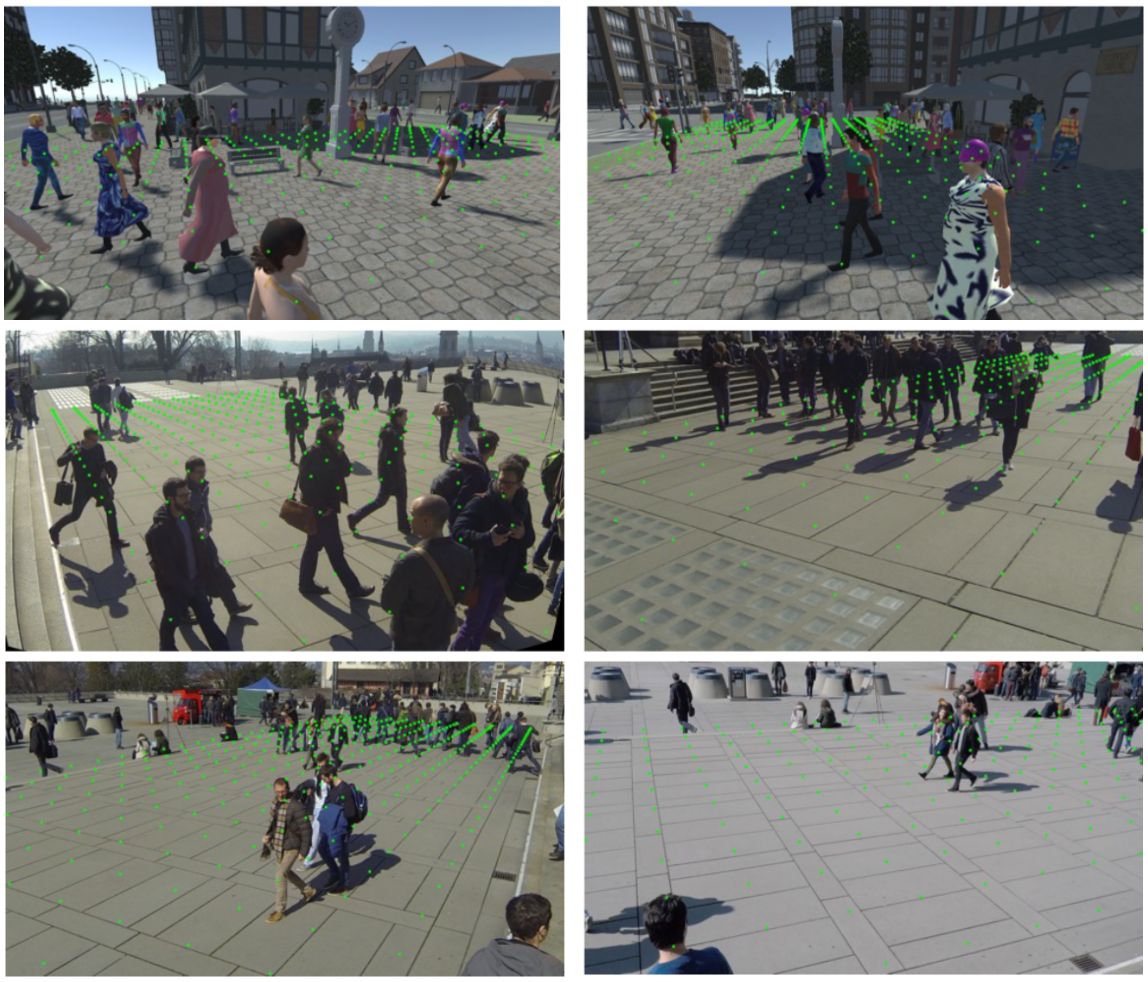


Fig 31. Example of the dataset calibration file verification

Furthermore, the distribution of pedestrians in each dataset is analysed using the *matplotlib* library and heatmap graphs are plotted for the 10% data split meant for testing. Fig 32 and Fig 33 demonstrate the results, where it can be noticed that the *MultiviewX* dataset has a more uniform distribution of pedestrians compared to the *WILDTRACK* dataset. The latter also has fewer hot spots, and they are more concentrated in a smaller area. Meanwhile, the MultiviewX heatmap indicates that the pedestrians are more spread out, with fewer outliers, thus providing a more consistent testing platform.

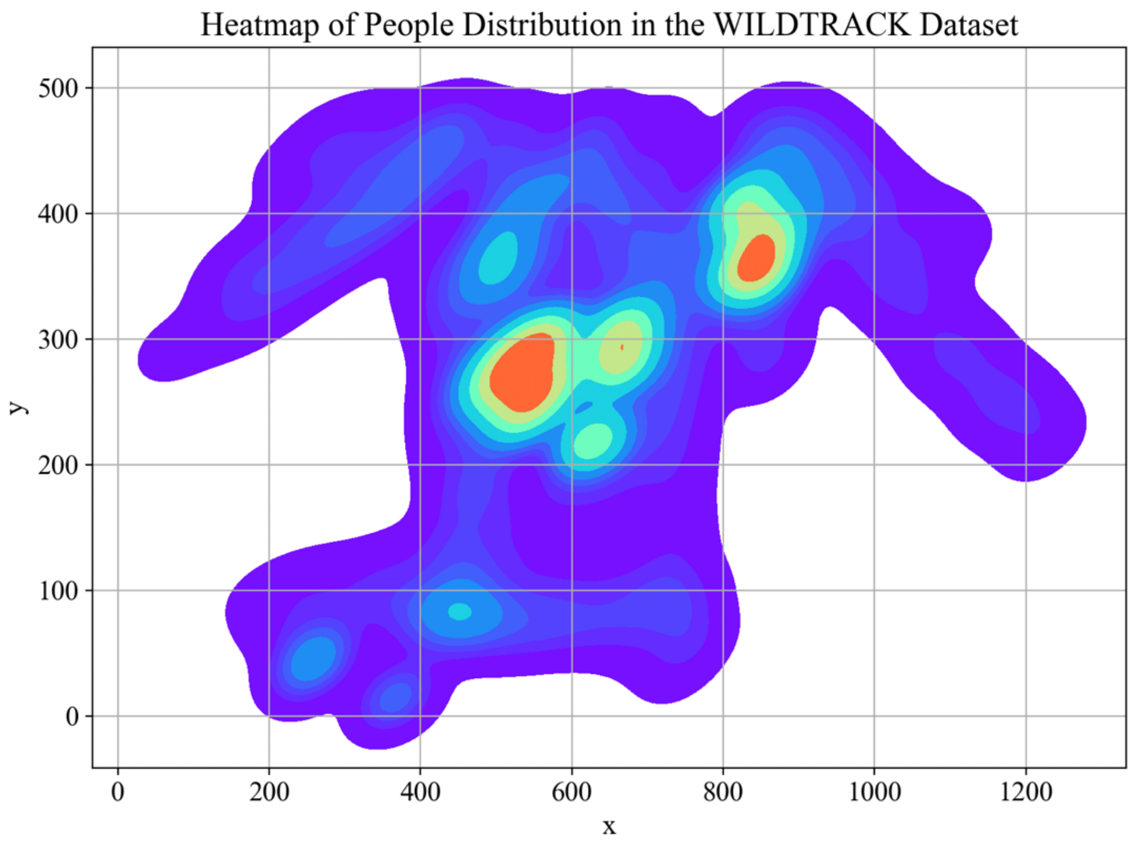


Fig 32. Heatmap of pedestrian distribution in the testing split of the *WILDTRACK* dataset

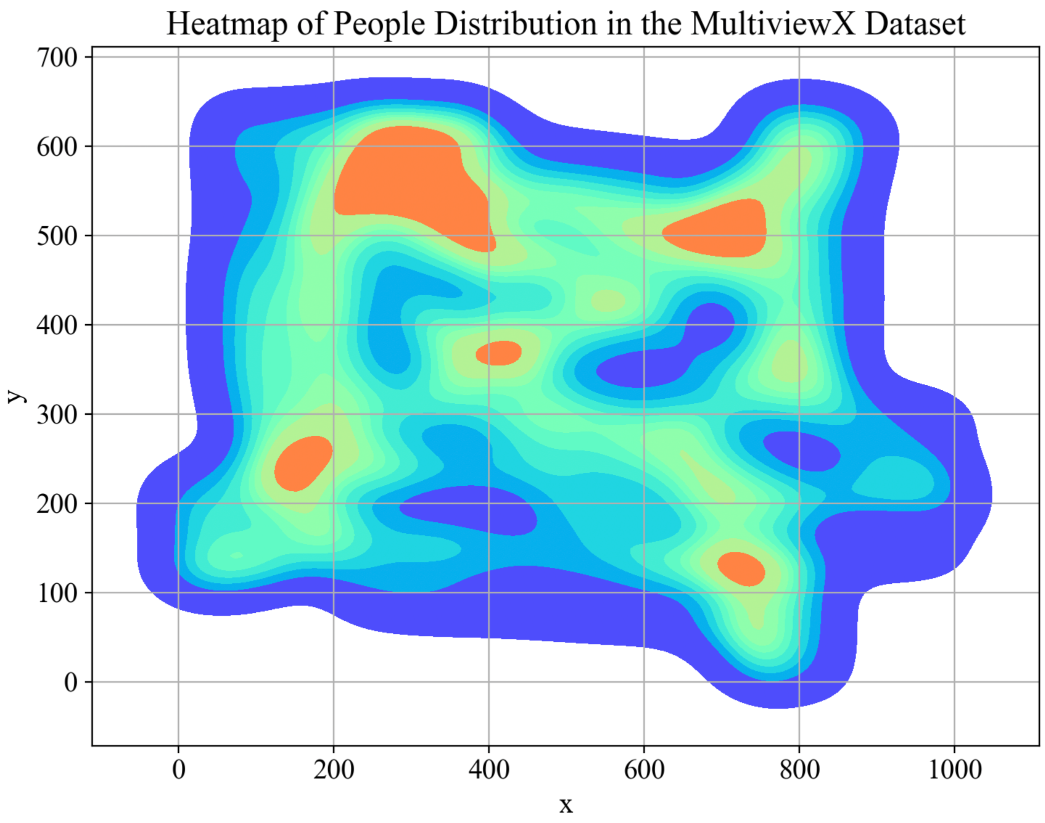


Fig 33. Heatmap of pedestrian distribution in the testing split of the *MultiviewX* dataset

3.3. Various Camera Configuration Training

The research aims to evaluate multi-view social distancing detection systems, requiring the training of a machine learning model variant for each combination of camera setup and encoder model. Fig 34 shows the average validation loss over training epochs for different camera counts, using *ResNet18* (the default encoder for the EarlyBird model) as the baseline.

The results demonstrate that more cameras generally lead to a lower initial validation loss, suggesting faster early learning and potentially better performance. Notably, increasing camera count from 2 to 3 resulted in a 45.8% reduction in initial loss. Subsequent camera additions yielded improvements, ranging from 12.7% to 23.5%. Overall, the validation loss for all configurations consistently decreases over the epochs, indicating successful training and ongoing improvement in model performance.

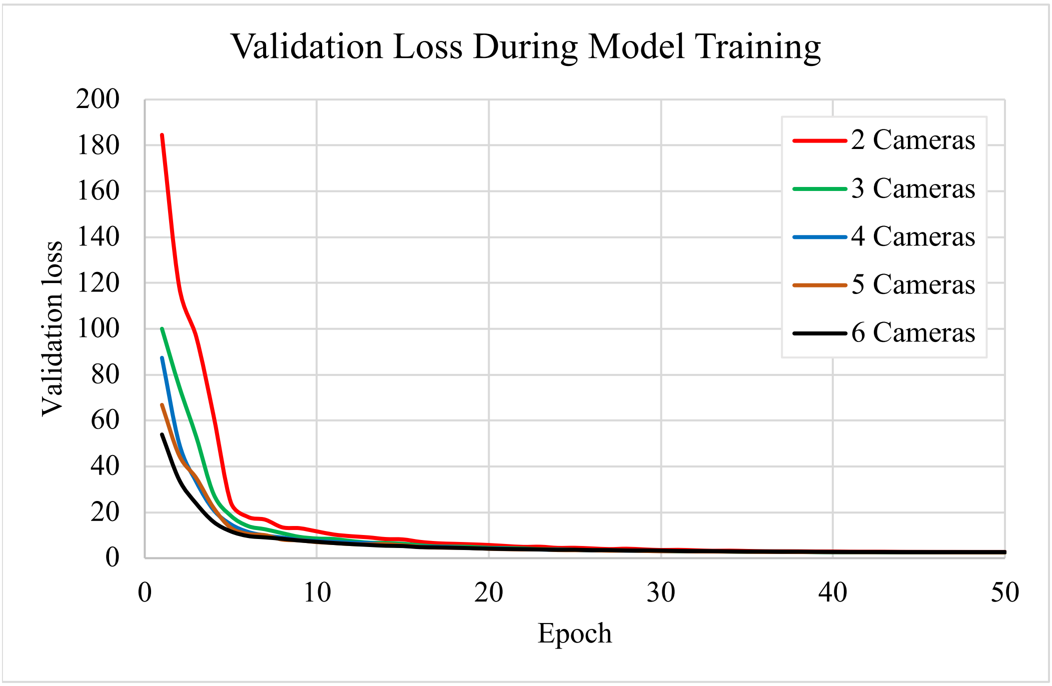


Fig 34. Validation loss progress during system model training on the *MultiviewX* dataset (encoder – *ResNet18*)

Table 2 presents the training times for various camera configurations and model variants. The modified model with *TinyNet-E* encoder consistently outperforms other variants, achieving similar results to the baseline model but with training durations reduced by 4.45% to 10.5% for 2-4 cameras and 9.81% to 16.94% for 5-7 cameras. This indicates a better model efficiency with the modified encoder type.

Alternatively, models with *TinyViT* and *Swin Transformer* encoders exhibit increased training times, peaking at 36.14% and 86.31% longer than the *ResNet18* variant, respectively. This confirms the theory that more complex encoder types require greater computational resources. Overall, training time increases by an average of 13.84% per additional camera for models with the *ResNet18* encoder, with similar increases noticeable in all model variants.

Table 2. Model training times for all evaluated encoder types

Camera count	Encoder type			
	ResNet18	**TinyViT**	**Swin Transformer**	**TinyNet-E**
2	4.27h	5.17h	5.46h	4.08h
3	4.93h	-	-	4.73h
4	5.81h	7.91h	9.52h	5.20h
5	6.42h	-	-	5.79h
6	7.38h	9.9h	13.75h	6.13h
7	8.15h	-	-	6.96h

The training progresses by iterating on the model weights and attempting to minimise the loss. Fig 35 illustrates the model outputs at various stages of training. Initial observations indicate the output is very noisy, with multiple high-intensity regions in the first epoch. Epoch 3 is slightly less noisy, with the model starting to learn and focus on specific areas. Epochs 20 and 50 indicate a much clearer output with well-defined and distinct high-intensity regions, confirming the model‘s ability to identify and focus on the relevant features in the input data.

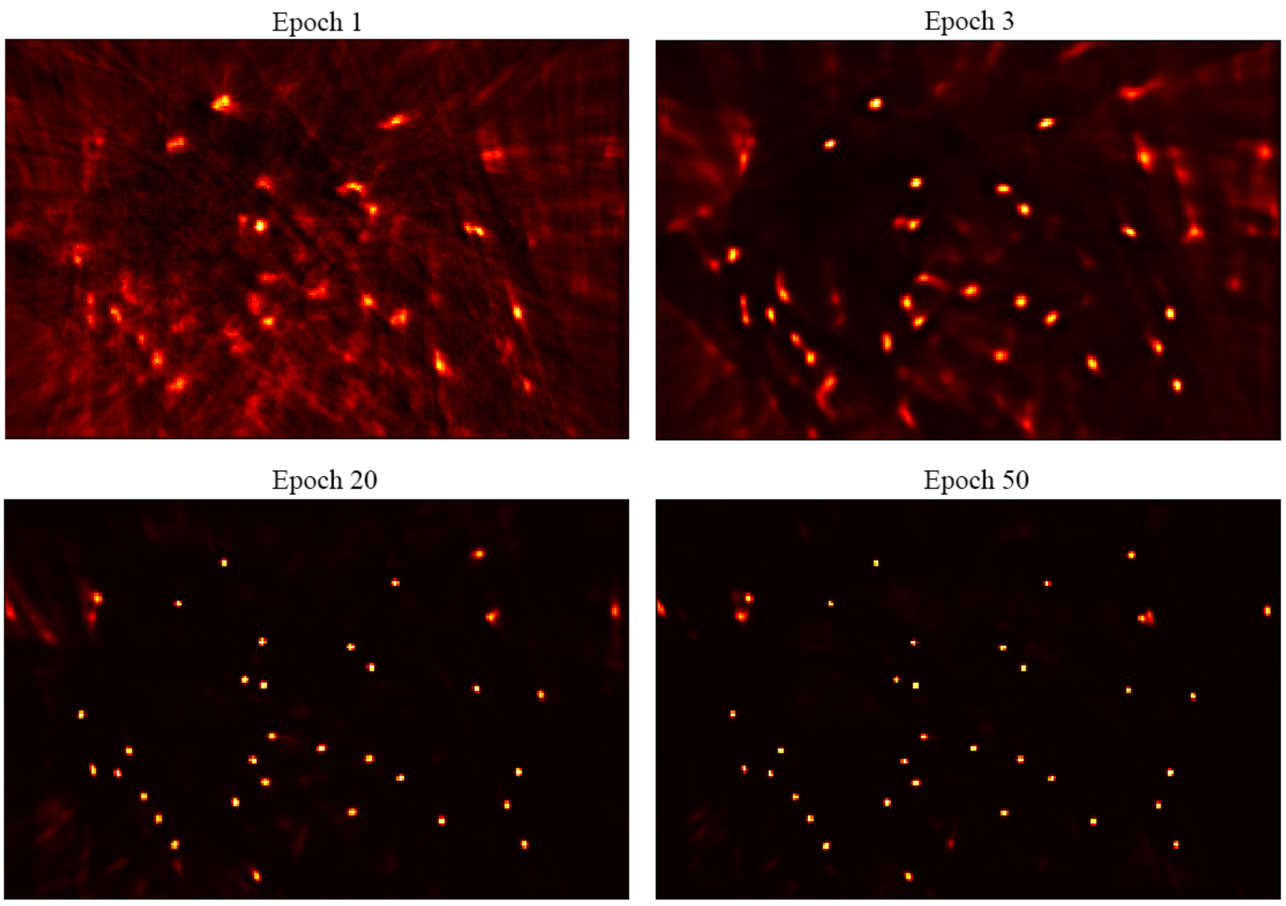


Fig 35. Model training progress images during various stages of training

3.4. Investigation of Camera Placement Impact on Social Distancing Detection Accuracy

This chapter examines how camera placement affects the accuracy of a multi-view social distancing detection system. Fig 36 provides an example of the system's prediction output graph. It indicates the red dots for each detected pedestrian, while the blue lines show the detected social distancing violation. These graphs are generated for each frame in the tested dataset portion and allow the user to observe the violations visually.

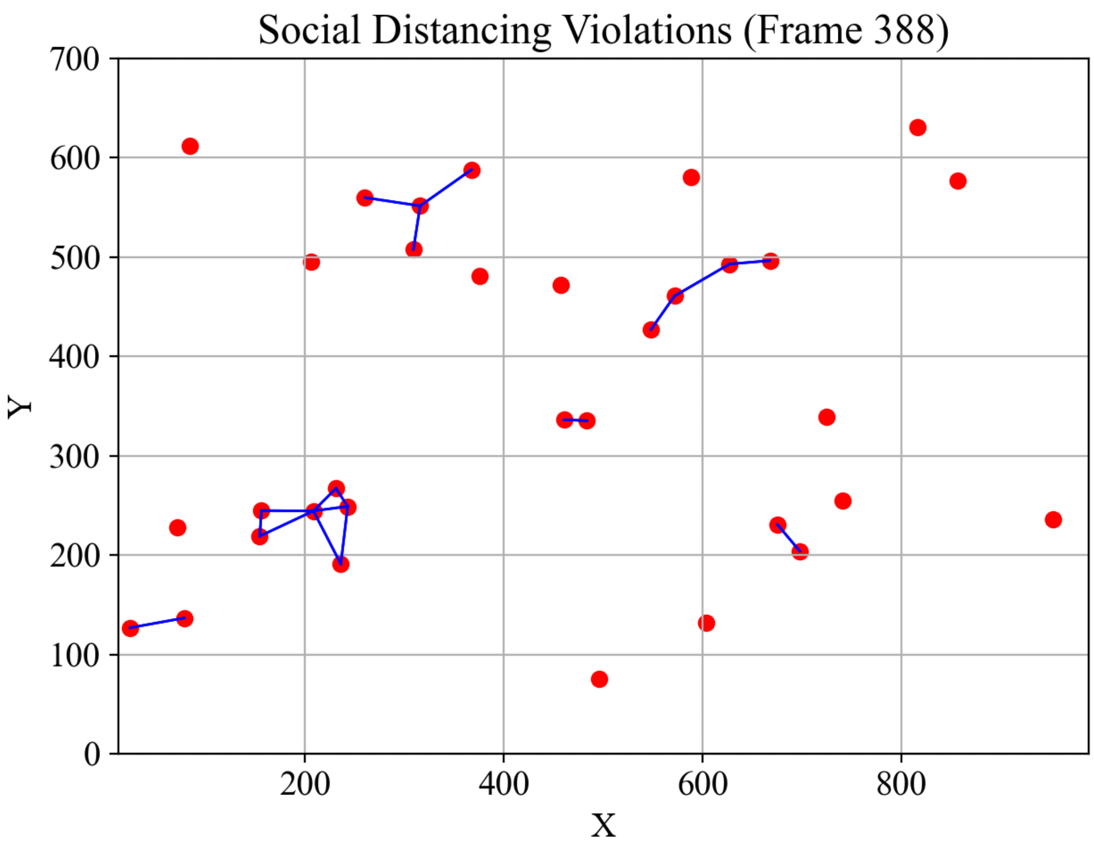


Fig 36. Example of the social distancing violations prediction on the *MultiviewX* dataset

Evaluation results on the *MultiviewX* dataset, provided in Table 3 and Fig 37, show that camera position matters, especially with fewer cameras. The most significant difference in accuracy (19.13%) between the same camera count configurations appears when only one camera is used. This difference reduces as more cameras are added, reaching just 1.61% with five cameras, suggesting that more cameras lead to better coverage, reducing the impact of their placements.

Looking at the best results for each setup, adding cameras clearly benefits accuracy. When moving from one to two cameras, accuracy increases by 16.53% and by another 12.82% when adding a third. However, further increases in camera count result in smaller gains, averaging 2.79% per additional camera. This indicates diminishing returns beyond three cameras. Lastly, adding more cameras slows the model down, with each additional camera increasing prediction time by an average of 12.24%. Overall, the results suggest that considering budget-bound applications, a correctly placed three-camera setup is sufficiently accurate in most applications. However, to achieve the best possible social distancing monitoring performance, adding more cameras will marginally improve the overall detection accuracy.

Table 3. Multi-view social distancing detection system evaluation results for different camera placement variants (*ResNet18* encoder, *MultiviewX* dataset)

Camera count	Configuration	MODA (%)	MODP (%)	Social Dist. F1	Tracking IDF1	Inference speed (ms)
1	A	54.42	82.98	0.528	0.482	-
	B	60.98	81.56	0.629	0.488	-
2	A	67.00	84.72	0.680	0.587	308
	B	73.03	85.93	0.733	0.603	314
3	A	85.54	87.34	0.806	0.722	364
	B	85.61	88.20	0.827	0.722	362
4	A	89.63	89.21	0.838	0.784	405
	B	88.89	90.26	0.861	0.799	408
5	A	92.64	90.55	0.872	0.809	454
	B	93.84	89.90	0.886	0.788	454
6	A	94.91	91.61	0.898	0.838	498

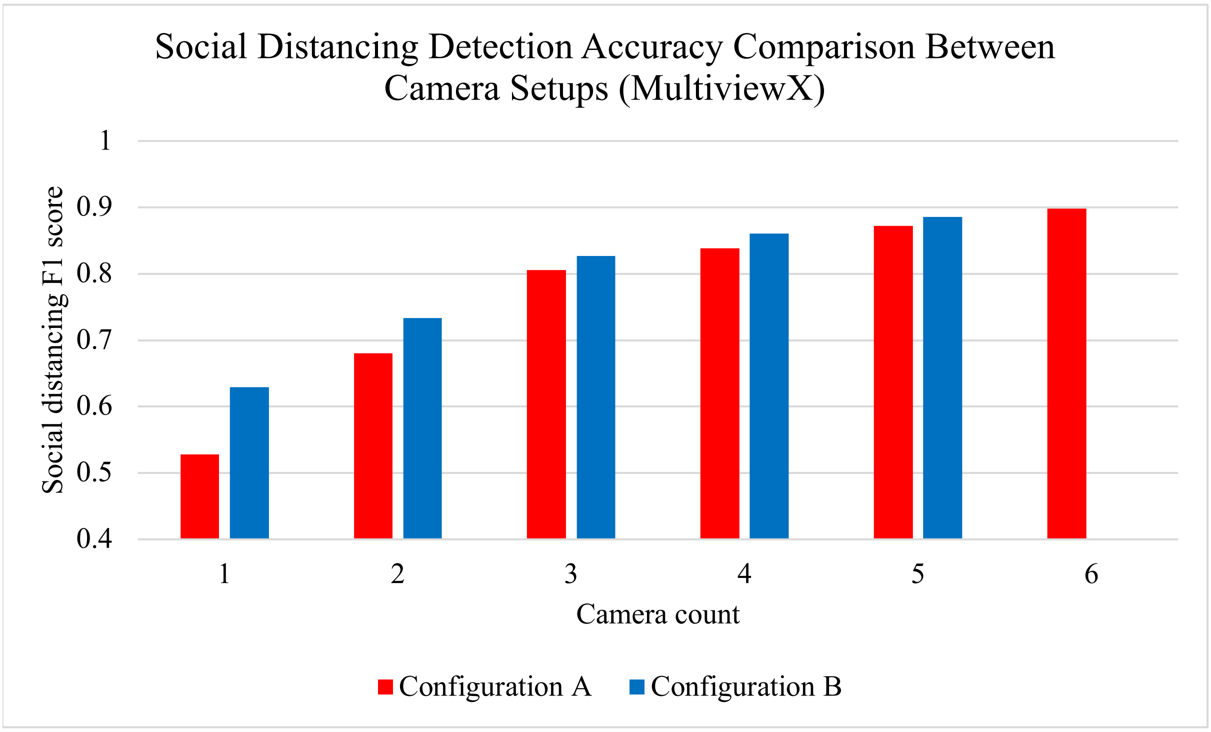


Fig 37. Social distancing detection F1 score results comparing different camera placements (*ResNet18* encoder, *MultiviewX* dataset)

Evaluation on the *WILDTRACK* dataset reveals similar trends but with a few differences. Notably, the most significant difference in social distancing detection accuracy between same-count camera setups occurs with four cameras (10.13%), along with a substantial 26.08% MODA difference. Focusing on each setup‘s best configuration, increasing camera count initially yields better gains: 7.2% from one to two cameras and 4.87% from two to three. However, subsequent additions provide less than 1% improvement. Inference time trends remain closely matched to the evaluation on the *MultiviewX* dataset.

Table 4. Multi-view social distancing detection system evaluation results for different camera placement variants (*ResNet18* encoder, *WILDTRACK* dataset)

Camera count	**Configuration**	**MODA (%)**	**MODP (%)**	**Social Dist. F1**	**Tracking IDF1**	**Inference speed (ms)**
1	A	65.76	76.42	0.761	0.599	-
	B	72.79	76.34	0.805	0.676	-
2	A	80.36	78.99	0.853	0.763	329
	B	84.24	80.17	0.863	0.816	330
3	A	90.44	80.24	0.899	0.887	383
	B	90.23	79.95	0.905	0.841	384
4	A	72.90	81.36	0.829	0.79	423
	B	91.91	80.69	0.913	0.839	426
5	A	85.92	80.46	0.895	0.882	470
	B	92.65	81.49	0.919	0.924	471
6	A	91.91	81.33	0.915	0.919	519
	B	91.91	81.78	0.920	0.914	513
7	A	91.07	81.90	0.912	0.934	565

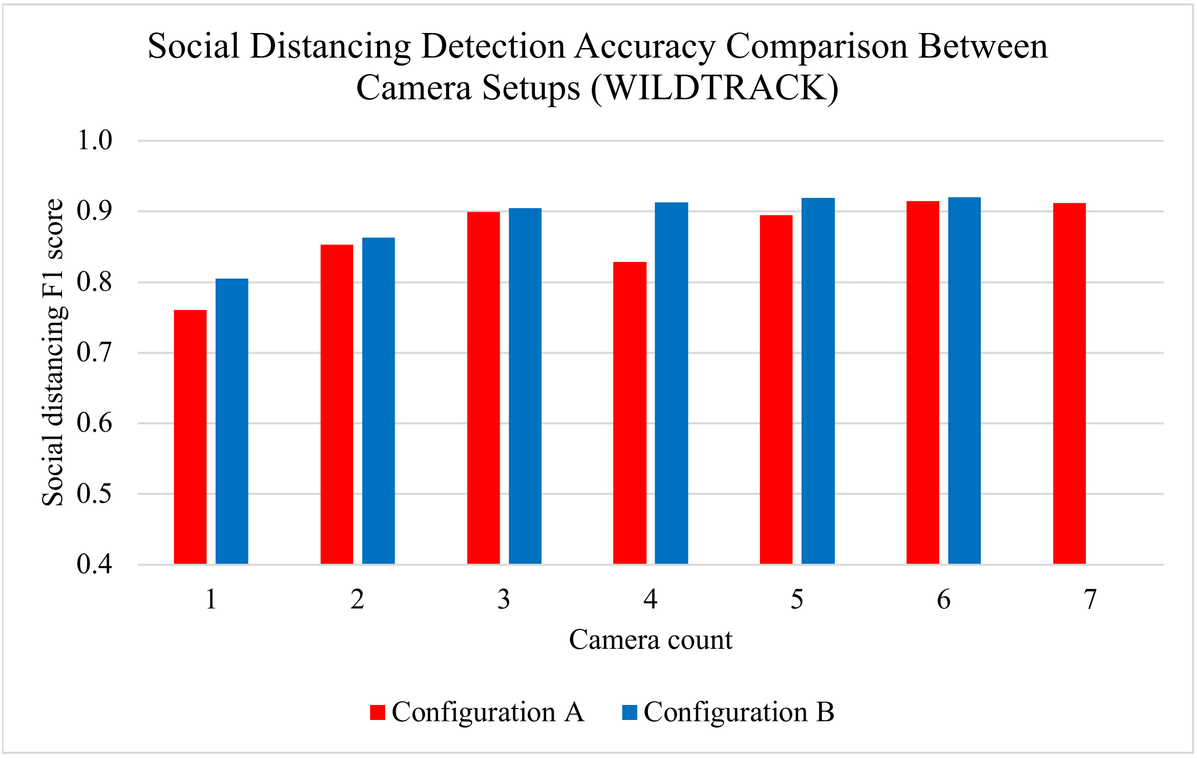


Fig 38. Social distancing detection F1 score results comparing different camera placements (*ResNet18* encoder, *WILDTRACK* dataset)

The *WILDTRACK* dataset provides real-world insight into the results. As provided in Fig 39, more people are walking together in groups. This is indicated by more clustered social distancing violations (highlighted with blue lines).

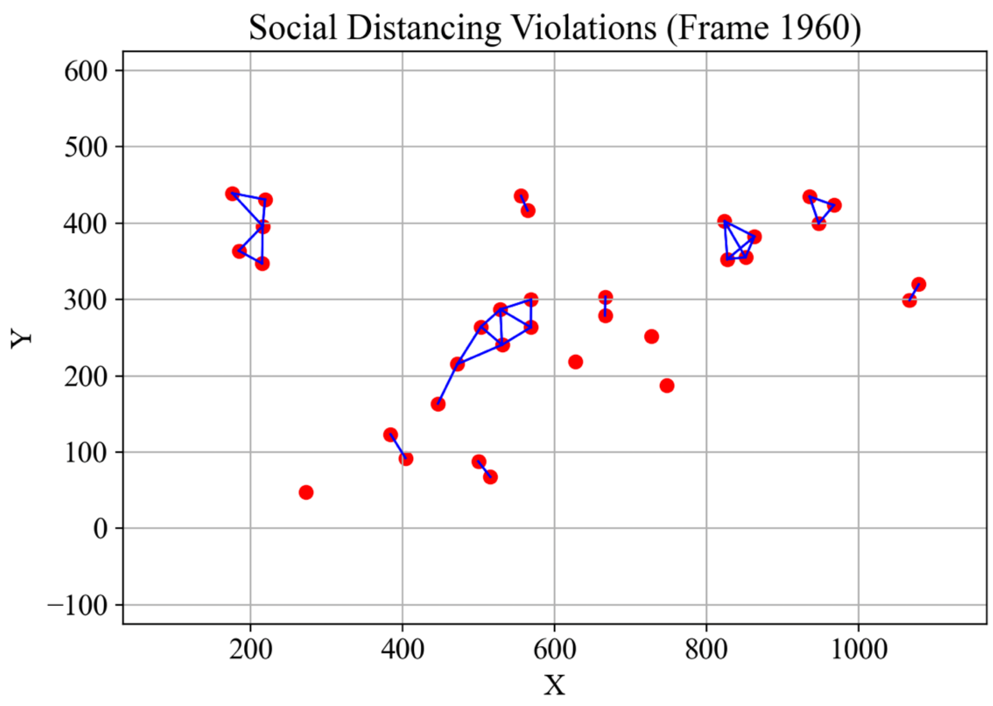


Fig 39. Example of the social distancing violations prediction on the *WILDTRACK* dataset

Supplementary analysis is performed as the main outlier from the *WILDTRACK* evaluation is the four-camera setup configuration *A*. Firstly, the model was retrained multiple times to reduce the possibility of any significant issues in the training process. However, the evaluation results remained the same, so the model output view was analysed next. In the provided view (Fig 40) it is evident that the selected camera configuration had multiple blind spots – areas where the cameras did not cover the scene. Additionally, the chosen configuration contains several cameras mounted lower than others, making them more susceptible to occlusion. Given these observations, it is evident that correct camera placement becomes more essential due to the *WILDTRACK* dataset's wider area and peculiar camera positions.

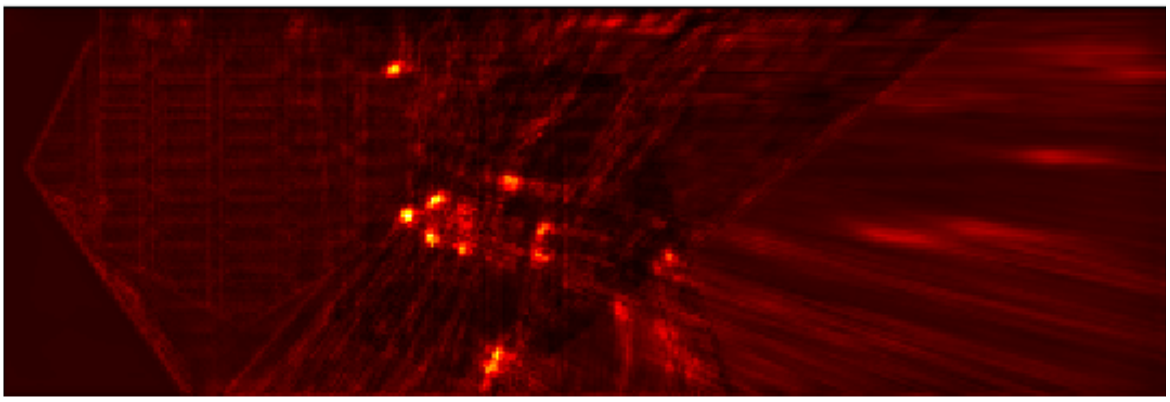


Fig 40. Model output view for the *WILDTRACK* dataset, using the 4x camera configuration *A*

3.5. Investigation of Complex Model Feature Extractor Impact on Accuracy

The performance comparison between *ResNet18*, *Swin Transformer* and *TinyViT* system encoders reveals additional insight for multi-view social distancing detection systems (Fig 41 and Table 5).

While MODP differences were negligible across all tested encoders, MODA and social distancing detection F1 scores displayed several trends.

The model with a *Swin Transformer* encoder consistently underperformed compared to base *ResNet18*, with decreases in MODA (2.04%-7.79%) and social distancing detection F1 score (3.67%-5.49%) across all tested camera counts. Additionally, this encoder exhibited a consistent decrease in tracking IDF1 score (8.62%-10.98%). Overall, the results suggest the *Swin Transformer's* incompatibility to be used in the system.

Conversely, using *TinyViT* as the encoder demonstrated promising results, with observable MODA increases ranging from 1.34% to 11.91% and social distancing detection F1 score improvements from 1.89% to 9.14% compared to the base *ResNet18* encoder. Although *TinyViT* shows slight decreases in tracking IDF1 scores at higher camera counts, its overall performance in other metrics suggests it is a viable alternative to the ResNet18 encoder for enhancing system accuracy.

Table 5. Multi-view social distancing detection system evaluation results for different encoder types (*MultiviewX* dataset)

Camera Count	**Encoder Type**	**MODA (%)**	**MODP (%)**	**Social Dist. F1**	**Tracking IDF1**	**Inference speed (ms)**
2	ResNet18	73.03	85.93	0.733	0.603	314
	Swin	67.34	84.66	0.698	0.551	409
	TinyViT	81.73	87.19	0.800	0.682	386
4	ResNet18	89.63	89.21	0.838	0.784	405
	Swin	85.61	86.89	0.792	0.705	603
	TinyViT	91.43	89.82	0.863	0.765	555
6	ResNet18	94.91	91.61	0.898	0.838	498
	Swin	92.97	89.44	0.865	0.746	790
	TinyViT	96.18	91.70	0.915	0.789	710

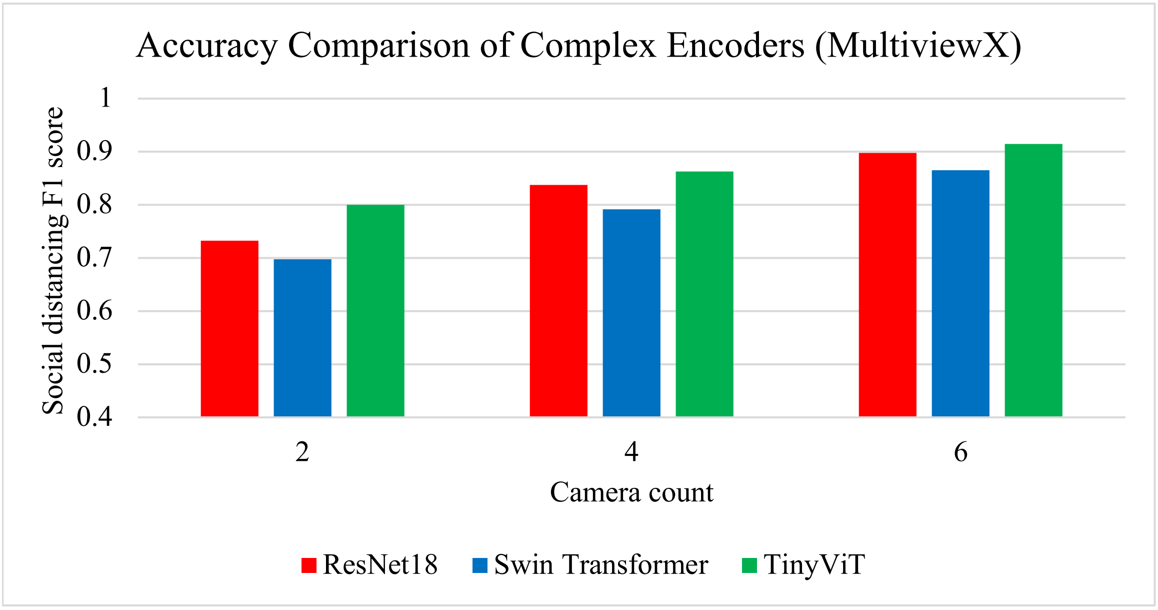


Fig 41. Social distancing detection F1 score results comparing *ResNet18*, *Swin* and *TinyViT* encoders (*MultiviewX*)

After evaluating the encoders on the *WILDTRACK* dataset (Fig 42 and Table 6), some differences are notable compared to the previous *MultiviewX* results. The model with the *Swin Transformer* encoder still underperforms compared to *ResNet18* in MODA (0.15%-4.84%) but shows a mixed impact on the social distancing detection F1 score. It slightly improves with a two-camera setup (0.59%) and marginally decreases with the four and six-camera configurations (2.63% and 0.87%, respectively). Regarding the tracking IDF1 score, it still shows a slight decrease across all camera counts.

TinyViT provides less positive results compared to its evaluation on the *MultiviewX* dataset. It improves MODA (1.83%) and social distancing detection scores (1.52%) with a two-camera configuration but fails to reach *ResNet18* performance levels on both metrics with four and six cameras. However, the *TinyViT* encoder increases the tracking IDF1 score by 8.52% with a two-camera setup, 11.92% with a four-camera setup, and 1.31% when using the six-camera configuration on the *WILDTRACK* dataset.

Table 6. Multi-view social distancing detection system evaluation results for different encoder types (*WILDTRACK* dataset)

Camera Count	**Encoder Type**	**MODA (%)**	**MODP (%)**	**Social Dist. F1**	**Tracking IDF1**	**Inference speed (ms)**
2	ResNet18	80.36	78.99	0.853	0.763	329
	Swin	76.47	79.06	0.858	0.751	428
	TinyViT	81.83	79.80	0.866	0.828	407
4	ResNet18	91.91	80.69	0.913	0.839	426
	Swin	91.77	79.59	0.889	0.905	611
	TinyViT	90.44	81.63	0.903	0.939	566
6	ResNet18	91.91	81.33	0.915	0.919	519
	Swin	91.49	79.85	0.907	0.900	804
	TinyViT	90.86	82.03	0.914	0.931	727

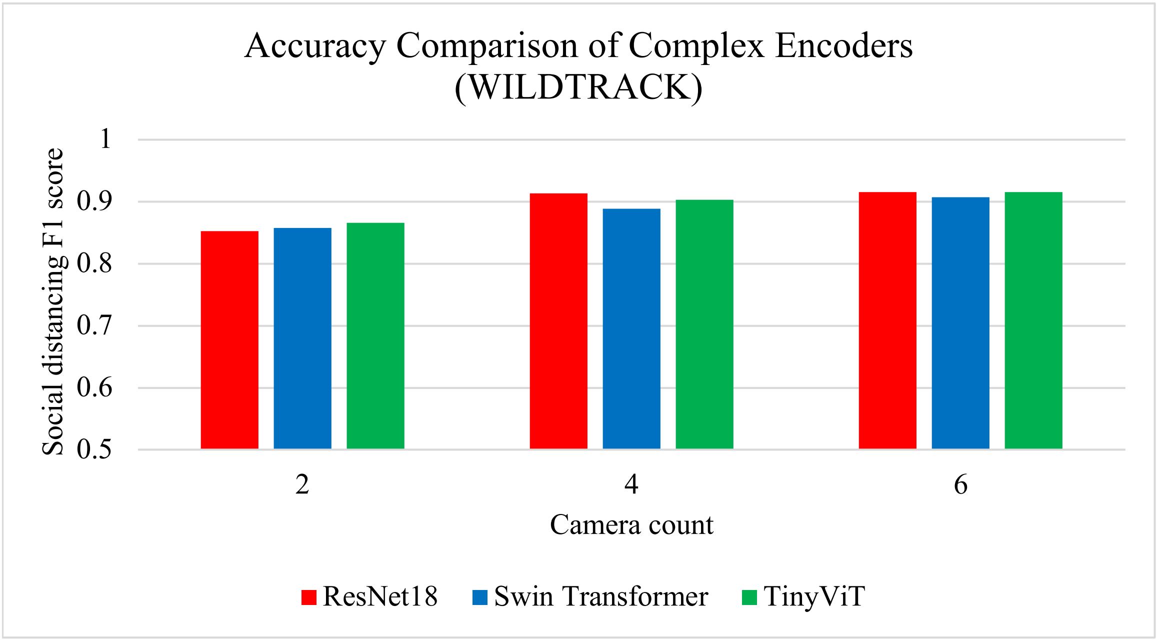


Fig 42. Social distancing detection F1 score results comparing ResNet18, Swin and TinyViT encoders (*WILDTRACK*)

The visualisation of inference time differences for all three model variants across the evaluated configurations is provided in Fig 43. Compared to *ResNet18*, *Swin Transformer* shows the most significant increases in inference time, with a 30.25% increase for two cameras, a 48.89% increase for four cameras, and a 58.63% increase for six cameras. Moreover, the multi-view social distancing detection system with a *TinyViT* encoder demonstrates a lesser increase in inference time compared to *ResNet18*, with a 22.93% increase for two cameras, 37.04% increase for four cameras, and 42.57% increase for six cameras. These findings suggest that more complex encoders may offer advantages in terms of accuracy, but they come at the cost of increased inference and training durations.

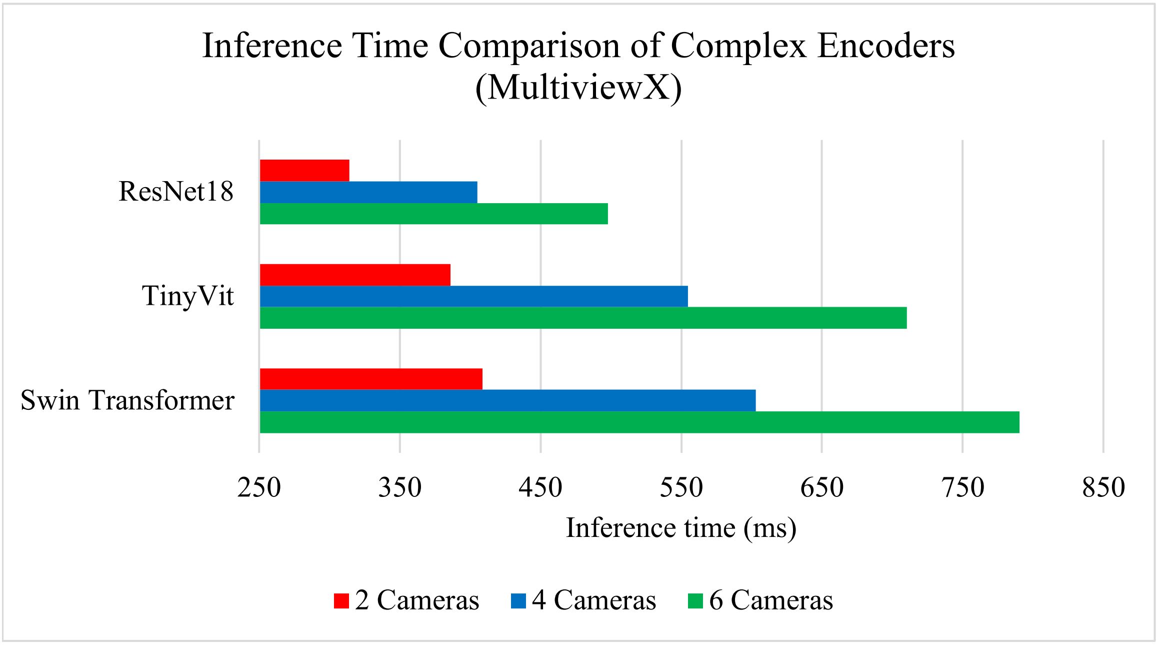


Fig 43. Inference time results comparing *ResNet18*, *Swin* and *TinyViT* encoders (*MultiviewX*)

3.6. Investigation of Using TinyNet-E as the Feature Extractor on Inference Speed

This work implements and evaluates a *TinyNet-E* encoder to enhance system efficiency as a replacement for the base *ResNet18* encoder. Fig 44 compares the model output views for both encoder variants. Notably, the feature maps produced by *TinyNet-E* differ from *ResNet18*, showing more high-intensity regions early in training. Despite this difference, the *TinyNet-E* model effectively highlights pedestrian features, enabling further evaluation of this implementation.

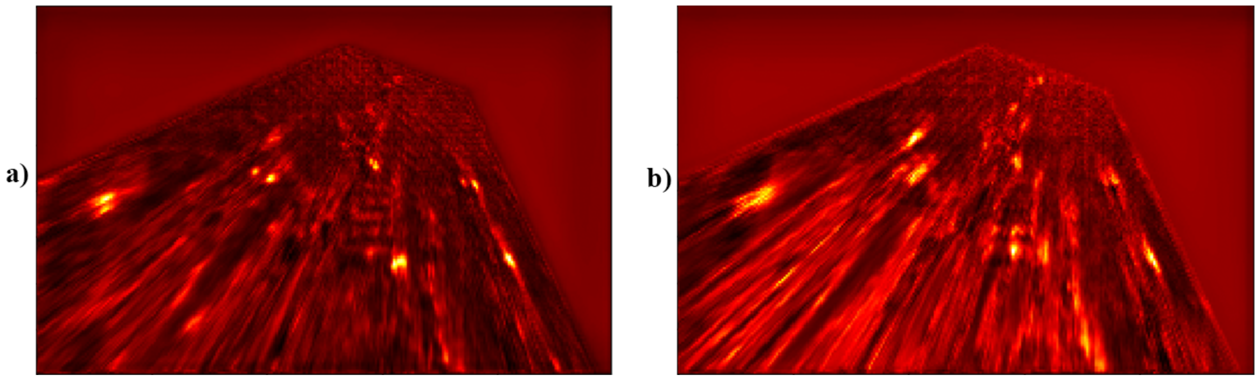


Fig 44. Training view from camera 3 of the MultiviewX dataset using a) ResNet18 and b) TinyNet as feature extractors (epoch – 1)

Evaluating the use of a *TinyNet-E* encoder against the base *ResNet18* on the *MultiviewX* dataset reveals mixed performance results. Regarding MODA and MODP, the *TinyNet-E* encoder demonstrates small decreases averaging 0.28% and 1.32%, respectively. A similar result is notable when evaluating the social distancing detection score, with an average reduction of 1.99%. However, the tracking suffers a noticeable loss in performance, averaging a 2.82% decrease in IDF1 score, with the highest drop (11.58%) evident when the six-camera configuration is used.

Table 7. Multi-view social distancing detection system evaluation results comparing *ResNet18* and *TinyNet* as encoders (*MultiviewX* dataset)

Encoder Type	**Camera Count**	**MODA (%)**	**MODP (%)**	**Social Dist. F1**	**Tracking IDF1**	**Inference speed (ms)**
ResNet18	1	60.98	81.56	0.629	0.488	-
	2	73.03	85.93	0.733	0.603	314
	3	85.61	88.20	0.827	0.722	362
	4	88.89	90.26	0.861	0.799	408
	5	93.84	89.90	0.886	0.788	454
	6	94.91	91.61	0.898	0.838	498
TinyNet-E	1	60.64	81.13	0.624	0.505	-
	2	78.71	84.76	0.735	0.629	295
	3	83	87.4	0.813	0.682	331
	4	89.63	88.53	0.842	0.757	364
	5	92.57	89.21	0.849	0.770	401
	6	93.04	89.41	0.868	0.741	436

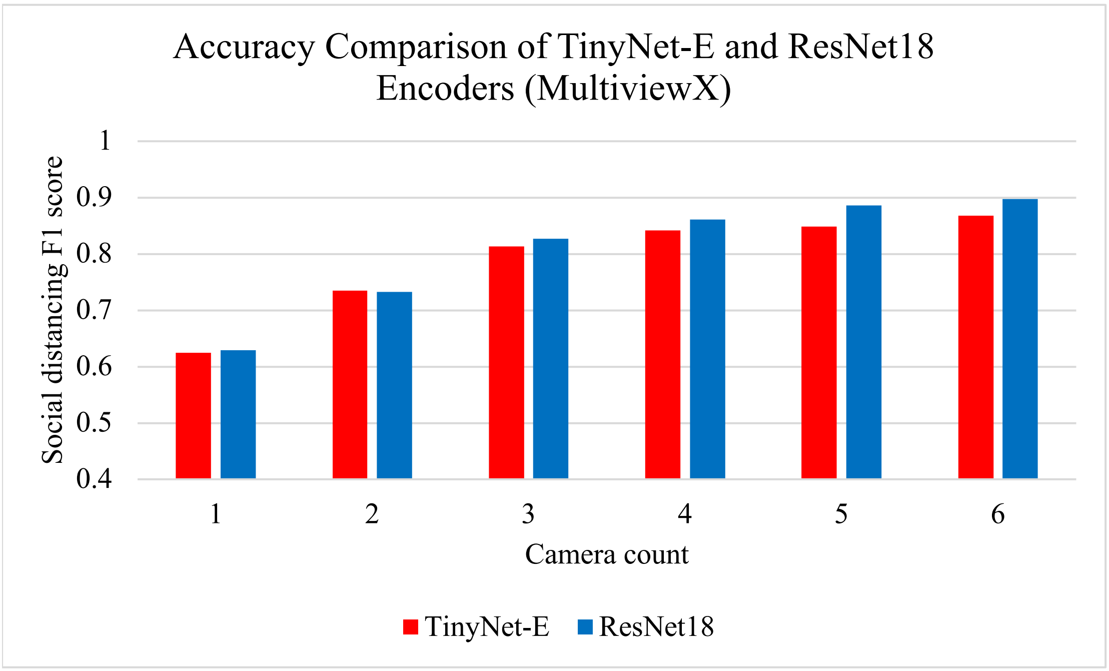


Fig 45. Social distancing detection F1 score results comparing *ResNet18* and *TinyNet* encoders (*MultiviewX*)

Executing the same evaluation process on the *WILDTRACK* dataset yields similar results (Fig 46 and Table 8), with the TinyNet-E encoder averaging decreases of 0.17% and 0.38% for MODA and

MODP, respectively. Social distancing detection F1 score also reports the *TinyNet-E* encoder use to cause an average drop of 0.34% with the highest difference of 1.74% appearing when using five cameras. Regarding the tracking IDF1 score, contrary to *MultiviewX* results, implementing a *TinyNet-E* encoder in the system improves the tracking performance by an average increase of 2.53%.

Table 8. Multi-view social distancing detection system evaluation results comparing *ResNet18* and *TinyNet* as encoders (*WILDTRACK* dataset)

Encoder Type	**Camera Count**	**MODA (%)**	**MODP (%)**	**Social Dist. F1**	**Tracking IDF1**	**Inference speed (ms)**
ResNet18	1	72.79	76.34	0.805	0.676	-
	2	84.24	80.17	0.863	0.816	330
	3	90.23	79.95	0.905	0.841	383
	4	91.91	80.69	0.913	0.839	426
	5	92.65	81.49	0.919	0.924	471
	6	91.91	81.78	0.920	0.914	513
	7	91.07	81.90	0.912	0.934	565
TinyNet-E	1	73.63	76.87	0.808	0.724	-
	2	87.08	80.38	0.875	0.867	308
	3	89.39	81.33	0.898	0.817	322
	4	89.71	80.65	0.899	0.893	360
	5	91.7	80.13	0.903	0.939	423
	6	90.97	80.82	0.914	0.913	455
	7	90.76	79.89	0.917	0.927	490

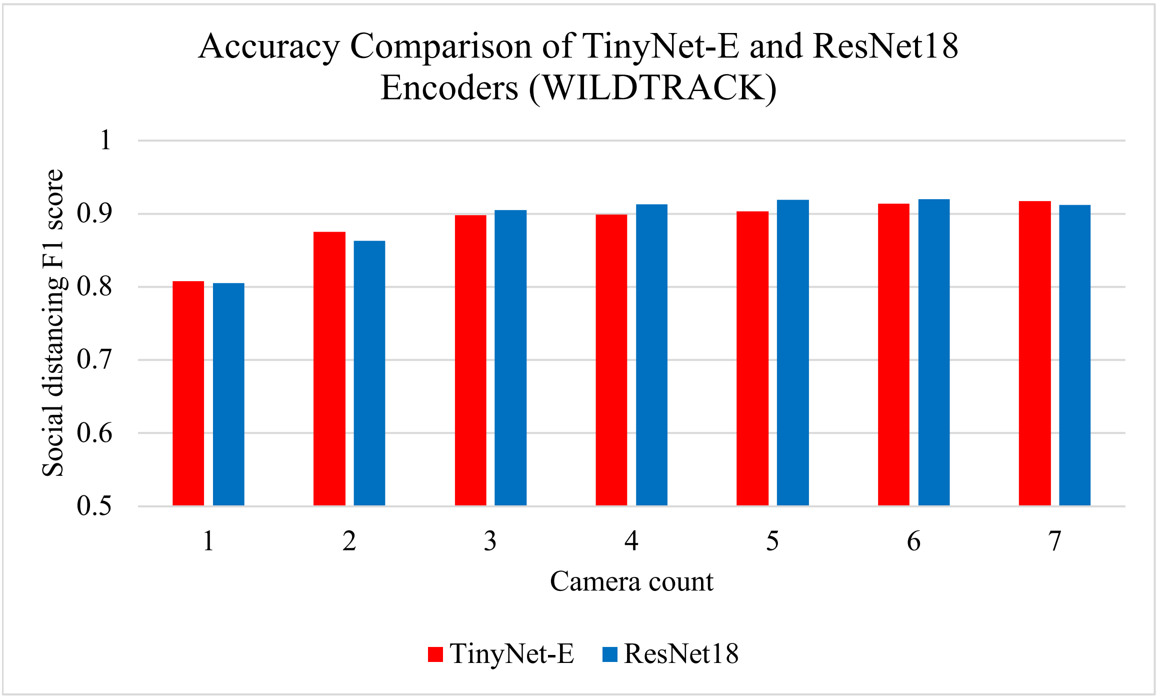


Fig 46. Social distancing detection F1 score results comparing *ResNet18* and *TinyNet* encoders (*WILDTRACK*)

The primary purpose of modifying the existing system to utilise a *TinyNet-E* encoder is to achieve better efficiency and inference speeds. Analysing the result data (Fig 47, Table 7 and Table 8), it is evident that this modification greatly reduces the overall prediction times. Considering data from the *WILDTRACK* evaluation (due to more samples), improvements across all camera configurations are made compared to the base model with a *ResNet18* encoder. Notably, inference time was reduced by 6.67% for a single-camera setup. Two and three-camera configurations decreased 15.93% and 15.49%, respectively. Meanwhile, systems with 5 to 7 cameras achieved an average reduction of 11.59%.

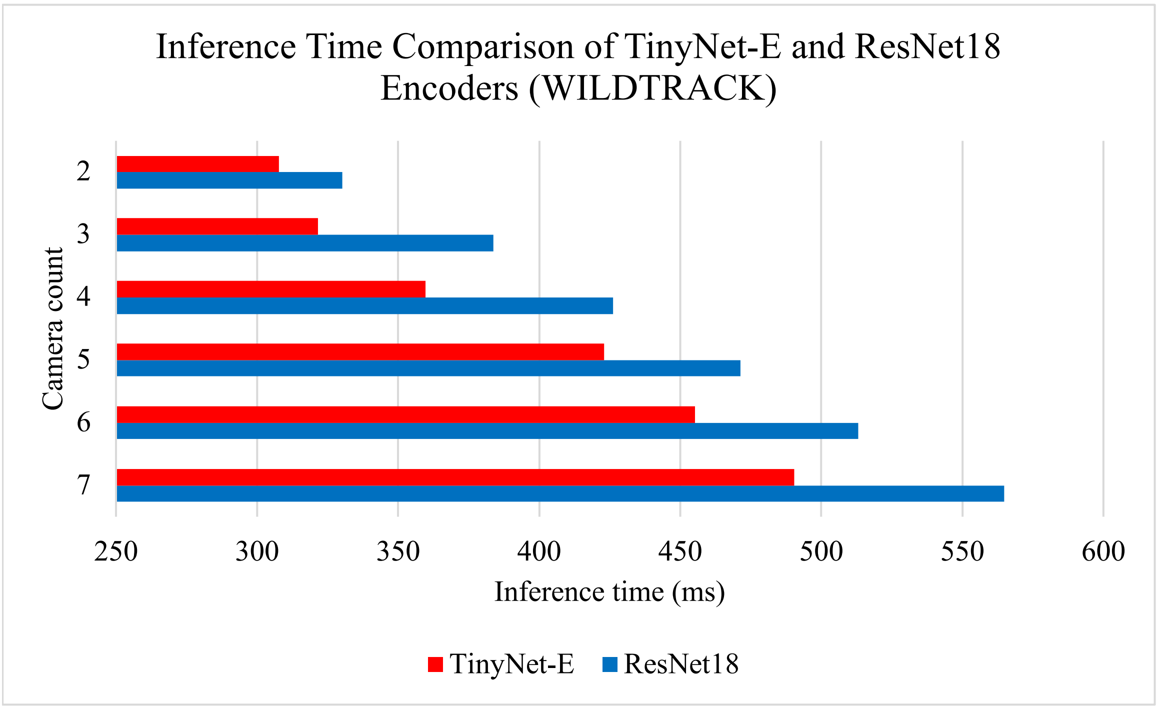


Fig 47. Inference time results for varied camera placements comparing *TinyNet* and *ResNet18* encoders

Conclusions

After performing the investigation of social distancing detection systems, the following conclusions are made:

1. Reviewing existing social distancing detection systems identified their main fundamental limitation: occlusion, as they use only the video feed of a single camera. Additionally, the accuracy of such systems highly depends on the placement of the camera, as they usually have to capture a large area.
2. After identifying the main limitations of existing social distancing detection systems, a new solution was proposed that uses multiple cameras capturing the same area from different angles and positions. The implementation was done using *Python* as the primary programming language, with the *PyTorch* library as the main machine learning interface. A state-of-the-art model, *EarlyBird*, was used as the system's base machine learning stage and modified for better evaluation capabilities to achieve the desired results.
3. The initial evaluation results revealed promising results, as using even two cameras instead of a single camera yields an increase in social distancing detection score of 16.53% on the *MultiviewX* dataset and a rise of 7.2% on the *WILDTRACK* dataset. Adding more cameras helps to get even better results with noticeable improvements in accuracy. Comparing the traditional single-view social distancing detection systems performance to a six-camera setup on the *MultiviewX* and a seven-camera setup on the *WILDTRACK* datasets, we see improved scores by 70.08% and 13.29%, respectively. The difference between these results proves that the performance of any social distancing monitoring system is partly dependent on external factors as well. This was also noticed in the evaluation progress, as the difference between same-count camera setups once reached 10.13%, thus confirming that camera placement is a crucial step in setting up a social distancing detection system.
4. Accuracy improvements have been achieved by implementing a more complex encoder in the early stage to extract more detailed feature maps. Using a *Swin Transformer* as the encoder reduced social distancing violation detection accuracy by up to 5.49%. However, using the *TinyViT* model instead of the base *ResNet18* encoder produced accuracy score increases of up to 9.14% and MODA result improvements of 11.91%. These results were achieved on the *MultiviewX* dataset, and evaluating the changes on the *WILDTRACK* dataset, the scores were almost identical, with the Swin Transformer again, slightly underperforming. Regarding efficiency, the modified systems with *Swin Transformer* and *TinyViT* encoders took significantly longer to provide a final prediction output, with six-camera setup inference time increases of 58.63% and 42.57%, respectively.
5. The use of a *TinyNet-E* model as the encoder improved the efficiency of the final social distancing detection system. Implementing the less complex model reduced the social distancing detection score by an average of 1.99%. However, the inference times of the modified system decreased by an average of 15.49%. This demonstrates that such a system is a viable alternative in budget-bound applications, as the reduction in required computational resources outweighs the lessened social distancing detection accuracy.

List of references

1. DAS, Sreetama. Anirban NAG. Dhruba ADHIKARY. et al. *Computer Vision-based Social Distancing Surveillance Solution with Optional Automated Camera Calibration for Large Scale Deployment* [interactive]. arXiv, 2021, [accessed 2024-05-24]. Available at: http://arxiv.org/abs/2104.10891.

2. SHAH, Juhi. Mahavir CHANDALIYA. Harsh BHUTA. et al. Social Distancing Detection Using Computer Vision. In *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)* [interactive]. 2021, p. 1359–1365. [accessed 2024-05-26]. Available at: https://ieeexplore.ieee.org/document/9418312.

3. WEISSTEIN, Eric W. Convolution. In [interactive]. [accessed 2024-05-26]. Available at: https://mathworld.wolfram.com/.

4. HEALY, Timothy J. Convolution revisited. In *IEEE Spectrum*. 1969, Vol. 6, no. 4, p. 87–93.

5. SUMNE, D. B. The Convolution Transform. By Hirschmann & Widder . Pp. x + 268. 45s. 1955. (Princeton University Press. London : Cumberlege). In *The Mathematical Gazette*. 1957, Vol. 41, no. 335, p. 71–72.

6. BOEHME, T. K. and Ron BRACEWELL. The Fourier Transform and its Applications. In *The American Mathematical Monthly* [interactive]. 1966, p. 685. [accessed 2024-05-26]. Available at: https://www.jstor.org/stable/2314845?origin=crossref.

7. MARCO, Simone DE. Minh-Duc HUA. Robert MAHONY. et al. Homography Estimation of a Moving Planar Scene From Direct Point Correspondence. In *IEEE Transactions on Control Systems Technology*. 2021, Vol. 29, no. 3, p. 1284–1295.

8. ZHAN, Xinrui. Yueran LIU. Jianke ZHU. et al. *Homography Decomposition Networks for Planar Object Tracking* [interactive]. arXiv, 2022, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/2112.07909.

9. NISHIDA, Kenji. Jun FUJIKI. Chikao TSUCHIYA. et al. Road Plane Detection using Differential Homography Estimated by Pair Feature Matching of Local Regions. In *Signal Processing, Pattern Recognition, and Applications / 722: Computer Graphics and Imaging* [interactive]. ACTAPRESS, 2011, [accessed 2024-05-26]. Available at: http://www.actapress.com/PaperInfo.aspx?paperId=451597.

10. MILSTEIN, Adam. Occupancy Grid Maps for Localization and Mapping. In JING, X.-J.Ed. [interactive]. InTech, 2008, [accessed 2024-05-26]. Available at: http://www.intechopen.com/books/motion_planning/occupancy_grid_maps_for_localization_and_mapping.

11. KUHN, Harold W. A tale of three eras: The discovery and rediscovery of the Hungarian Method. In *European Journal of Operational Research*. 2012, Vol. 219, no. 3, p. 641–651.

12. PASZKE, Adam. Sam GROSS. Francisco MASSA. et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library* [interactive]. arXiv, 2019, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/1912.01703.

13. CULJAK, Ivan. David ABRAM. Tomislav PRIBANIC. et al. A brief introduction to OpenCV. In *2012 Proceedings of the 35th International Convention MIPRO* [interactive]. 2012, p. 1725–1730. [accessed 2024-05-26]. Available at: https://ieeexplore.ieee.org/document/6240859.

14. DALTON, J. and A. DESHMANE. Artificial neural networks. In *IEEE Potentials*. 1991, Vol. 10, no. 2, p. 33–36.

15. JAIN, A.K. JIANCHANG MAO and K.M. MOHIUDDIN. Artificial neural networks: a tutorial. In *Computer*. 1996, Vol. 29, no. 3, p. 31–44.

16. YU, Tong. and Hong ZHU. *Hyper-Parameter Optimization: A Review of Algorithms and Applications* [interactive]. arXiv, 2020, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/2003.05689.

17. ALBAWI, Saad. Tareq Abed MOHAMMED. and Saad AL-ZAWI. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)* [interactive]. 2017, p. 1–6. [accessed 2024-05-26]. Available at: https://ieeexplore.ieee.org/document/8308186.

18. YAMASHITA, Rikiya. Mizuho NISHIO. Richard Kinh Gian DO. et al. Convolutional neural networks: an overview and application in radiology. In *Insights into Imaging*. 2018, Vol. 9, no. 4, p. 611–629.

19. GU, Jiuxiang. Zhenhua WANG. Jason KUEN. et al. *Recent Advances in Convolutional Neural Networks* [interactive]. arXiv, 2017, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/1512.07108.

20. GIRSHICK, Ross. Jeff DONAHUE. Trevor DARRELL. et al. *Rich feature hierarchies for accurate object detection and semantic segmentation* [interactive]. arXiv, 2014, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/1311.2524.

21. GIRSHICK, Ross. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)* [interactive]. 2015, p. 1440–1448. [accessed 2024-05-26]. Available at: https://ieeexplore.ieee.org/document/7410526.

22. REN, Shaoqing. Kaiming HE. Ross GIRSHICK. et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* [interactive]. arXiv, 2016, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/1506.01497.

23. DENG, Jia. Wei DONG. Richard SOCHER. et al. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition* [interactive]. 2009, p. 248–255. [accessed 2024-05-26]. Available at: https://ieeexplore.ieee.org/document/5206848.

24. KRIZHEVSKY, Alex. Ilya SUTSKEVER. and Geoffrey E HINTON. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems* [interactive]. Curran Associates, Inc., 2012, [accessed 2024-05-26]. Available at: https://proceedings.neurips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

25. SZEGEDY, Christian. Wei LIU. Yangqing JIA. et al. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [interactive]. 2015, p. 1–9. [accessed 2024-05-26]. Available at: https://ieeexplore.ieee.org/document/7298594.

26. HE, Kaiming. Xiangyu ZHANG. Shaoqing REN. et al. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [interactive]. 2016, p. 770–778. [accessed 2024-05-26]. Available at: https://ieeexplore.ieee.org/document/7780459.

27. CRAWFORD, Kate. and Trevor PAGLEN. Excavating AI: The Politics of Training Sets for Machine Learning. In - [interactive]. 2019, [accessed 2024-05-26]. Available at: https://excavating.ai.

28. DASHA.AI. The Environmental Impact of Training Models Like ChatGPT. In [interactive]. 2023, [accessed 2024-05-26]. Available at: https://dasha.ai/en-us/blog/the-environmental-impact-of-training-models-like-chatgpt.

29. HANDALAGE, Upulie. and Lakshini KUGANANDAMURTHY. *Real-Time Object Detection Using YOLO: A Review*. 2021, .

30. MELEK, Ceren Gulra. Elena Battini SONMEZ. and Songul ALBAYRAK. Object Detection in Shelf Images with YOLO. In *IEEE EUROCON 2019 -18th International Conference on Smart Technologies* [interactive]. 2019, p. 1–5. [accessed 2024-05-26]. Available at: https://ieeexplore.ieee.org/document/8861817.

31. BOCHKOVSKIY, Alexey. Chien-Yao WANG. and Hong-Yuan Mark LIAO. *YOLOv4: Optimal Speed and Accuracy of Object Detection* [interactive]. arXiv, 2020, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/2004.10934.

32. HAN, Kai. Yunhe WANG. Qiulin ZHANG. et al. *Model Rubik's Cube: Twisting Resolution, Depth and Width for TinyNets* [interactive]. arXiv, 2020, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/2010.14819.

33. HOU, Yunzhong. Liang ZHENG. and Stephen GOULD. *Multiview Detection with Feature Perspective Transformation* [interactive]. arXiv, 2021, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/2007.07247.

34. GUERRERO-GÓMEZ-OLMEDO, Ricardo. Beatriz TORRE-JIMÉNEZ. Roberto LÓPEZ-SASTRE. et al. Extremely Overlapping Vehicle Counting. In PAREDES, R. - CARDOSO, J.S. - PARDO, X.M.Eds. *Pattern Recognition and Image Analysis*. Springer International Publishing, 2015, p. 423–431.

35. TEEPE, Torben. Philipp WOLTERS. Johannes GILG. et al. *EarlyBird: Early-Fusion for Multi-View Tracking in the Bird's Eye View* [interactive]. arXiv, 2023, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/2310.13350.

36. WORLD HEALTH ORGANIZATION. Advice for the public on COVID-19 – World Health Organization. In [interactive]. 2023, [accessed 2024-05-26]. Available at: https://www.who.int/emergencies/diseases/novel-coronavirus-2019/advice-for-public.

37. HARTLEY, Richard. and Andrew ZISSERMAN. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003, 676 p. ISBN 978-0-521-54051-3.

38. NALAIE, Keivan. and Rong ZHENG. *Learning Online Policies for Person Tracking in Multi-View Environments* [interactive]. arXiv, 2023, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/2312.15858.

39. LUITEN, Jonathon. Aljosa OSEP. Patrick DENDORFER. et al. HOTA: A Higher Order Metric for Evaluating Multi-Object Tracking. In *International Journal of Computer Vision*. 2021, Vol. 129, no. 2, p. 548–578.

40. DECI. The Correct Way to Measure Inference Time of Deep Neural Networks. In *Deci* [interactive]. 2023, [accessed 2024-05-26]. Available at: https://deci.ai/blog/measure-inference-time-deep-neural-networks/.

41. GOUTTE, Cyril. and Eric GAUSSIER. A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation. In LOSADA, D.E. - FERNÁNDEZ-LUNA, J.M.Eds. *Advances in Information Retrieval*. Springer, 2005, p. 345–359.

42. WIGHTMAN, Ross. Nathan RAW. Alexander SOARE. et al. rwightman/pytorch-image-models: v0.8.10dev0 Release. In [interactive]. Zenodo, 2023 [accessed 2024-05-26]. Available at: https://zenodo.org/records/7618837.

43. LIBERTI, Leo. Carlile LAVOR. Nelson MACULAN. et al. *Euclidean distance geometry and applications* [interactive]. arXiv, 2012, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/1205.0349.

44. DHRUV, Akshit J. Reema PATEL. and Nishant DOSHI. Python: The Most Advanced Programming Language for Computer Science Applications: In *Proceedings of the International Conference on Culture Heritage, Education, Sustainable Tourism, and Innovation Technologies*. 2020, p. 292–299.

45. HARRIS, Charles R. K. Jarrod MILLMAN. Stéfan J. VAN DER WALT. et al. Array programming with NumPy. In *Nature*. 2020, Vol. 585, no. 7825, p. 357–362.

46. HUNTER, John D. Matplotlib: A 2D Graphics Environment. In *Computing in Science & Engineering*. 2007, Vol. 9, no. 3, p. 90–95.

47. ABADI, Martín. Paul BARHAM. Jianmin CHEN. et al. *TensorFlow: A system for large-scale machine learning* [interactive]. arXiv, 2016, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/1605.08695.

48. GOOGLE. Google Colaboratory. In [interactive]. [accessed 2024-05-26]. Available at: https://colab.research.google.com/.

49. CHAVDAROVA, Tatjana. Pierre BAQUÉ. Stéphane BOUQUET. et al. *The WILDTRACK Multi-Camera Person Dataset* [interactive]. arXiv, 2017, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/1707.09299.

List of digital resources

R1. DAS, Sreetama. Anirban NAG. Dhruba ADHIKARY. et al. Computer Vision-based Social Distancing Surveillance with Automated Camera Calibration for Large-scale Deployment. In *2021 IEEE 18th India Council International Conference (INDICON)* [interactive]. 2021, p. 1–6. [accessed 2024-05-26]. Available at: https://ieeexplore.ieee.org/document/9691485.

R2. SMITH, Steven W. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Pub., 1997, 626 p. ISBN 978-0-9660176-3-2.

R3. GUPTA, Prannaya. 2D Image Convolution with Numpy with a Handmade Sliding Window View. In *Medium* [interactive]. 2021, [accessed 2024-05-26]. Available at: https://medium.com/@thepyprogrammer/2d-image-convolution-with-numpy-with-a-handmade-sliding-window-view-946c4acb98b4.

R4. FISHER, Robert B. and Konstantinos KORYLLOS. Interactive Textbooks; Embedding Image Processing Operator Demonstrations in Text. In *International Journal of Pattern Recognition and Artificial Intelligence*. 1998, Vol. 12, no. 08, p. 1095–1123.

R5. SHANKAR, Yalda. Estimating a Homography Matrix. In *Medium* [interactive]. 2022, [accessed 2024-05-26]. Available at: https://towardsdatascience.com/estimating-a-homography-matrix-522c70ec4b2c.

R6. KANG, Lai. Yingmei WEI. Yuxiang XIE. et al. Combining Convolutional Neural Network and Photometric Refinement for Accurate Homography Estimation. In *IEEE Access*. 2019, Vol. PP, p. 1–1.

R7. BITTEL, Sebastian. Timo REHFELD. Michael WEBER. et al. Estimating high definition map parameters with convolutional neural networks. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* [interactive]. 2017, p. 52–56. [accessed 2024-05-26]. Available at: https://ieeexplore.ieee.org/document/8122577.

R8. OpenCV: Feature Matching. In *OpenCV* [interactive]. [accessed 2024-05-26]. Available at: https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html.

R9. ABDOLRASOL, Maher G. M. S. M. Suhail HUSSAIN. Taha Selim USTUN. et al. Artificial Neural Networks Based Optimization Techniques: A Review. In *Electronics*. 2021, Vol. 10, no. 21, p. 2689.

R10. KIM, Sung Eun. and Il Won SEO. Artificial Neural Network ensemble modeling with conjunctive data clustering for water quality prediction in rivers. In *Journal of Hydro-environment Research*. 2015, Vol. 9, no. 3, p. 325–339.

R11. SMITH, Leslie N. *A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay* [interactive]. arXiv, 2018, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/1803.09820.

R12. SAHA, Sumit. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. In *Medium* [interactive]. 2022, [accessed 2024-05-26]. Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

R13. FAROOQ, Umer. From R-CNN to Mask R-CNN. In *Medium* [interactive]. 2018, [accessed 2024-05-26]. Available at: https://medium.com/@umerfarooq_26378/from-r-cnn-to-mask-r-cnn-d6367b196cfd.

R14. REN, Shaoqing. Kaiming HE. Ross GIRSHICK. et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* [interactive]. arXiv, 2016, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/1506.01497.

R15. An Introduction to ImageNet. In *Roboflow Blog* [interactive]. 2021, [accessed 2024-05-26]. Available at: https://blog.roboflow.com/introduction-to-imagenet/.

R16. CHAUHAN, Nitin. Yolo Object Detection Made Easy. In *Medium* [interactive]. 2020, [accessed 2024-05-26]. Available at: https://medium.com/analytics-vidhya/yolo-object-detection-made-easy-7b17cc3e782f.

R17. HANDALAGE, Upulie. and Lakshini KUGANANDAMURTHY. *Real-Time Object Detection Using YOLO: A Review* [interactive]. 2021, Available at: https://www.academia.edu/download/67257544/Real_Time_Object_Detection_using_YOLO_A_review.pdf.

R18. HOU, Yunzhong. Liang ZHENG. and Stephen GOULD. *Multiview Detection with Feature Perspective Transformation* [interactive]. arXiv, 2021, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/2007.07247.

R19. TEEPE, Torben. Philipp WOLTERS. Johannes GILG. et al. *EarlyBird: Early-Fusion for Multi-View Tracking in the Bird's Eye View* [interactive]. arXiv, 2023, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/2310.13350.

R20. WANG, Ping. Lele HU. Guiyou LIU. et al. Prediction of Antimicrobial Peptides Based on Sequence Alignment and Feature Selection Methods. In *PLoS ONE*. 2011, Vol. 6, no. 4, p. e18476.

R21. CHAVDAROVA, Tatjana. Pierre BAQUÉ. Stéphane BOUQUET. et al. *The WILDTRACK Multi-Camera Person Dataset* [interactive]. arXiv, 2017, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/1707.09299.

R22. HOU, Yunzhong. Liang ZHENG. and Stephen GOULD. *Multiview Detection with Feature Perspective Transformation* [interactive]. arXiv, 2021, [accessed 2024-05-26]. Available at: http://arxiv.org/abs/2007.07247.

Appendices

Appendix 1. IVUS 2023 Conference Publication

A paper presented at the 28th IT conference IVUS 2023

Gliaubičiūtė D., Janavičius R., Gadeikytė A., Paulauskas L. (2023). *Influence of Aerial Image Resolution on Vehicle Detection Accuracy.*

Appendix 2. Camera Calibration Files Verification Results