University of Sheffield

Automating the tracking of individuals across multiple scenes/shots in video using a Deep Neural Network

Charalambos Georgiades

Supervisor: Dr Yoshi Gotoh

A report submitted in fulfilment of the requirements
for the degree of BSc in Computer Science

in the

Department of Computer Science

May 10, 2023

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Charalambos Georgiades

Signature: Charalambos Georgiades

Date: September 29, 2022

Abstract

Using computer vision to identify objects in a video stream is an extremely popular task being implemented by companies every day. Despite the popularity, developing an object detection model is still an extremely challenging and time-consuming task. Thus, the purpose of this project is to create an algorithm than can train an object detection model which is able to accurately locate and track any number of specified individuals through out a given video stream or image while simultaneously reducing the amount of manual effort needed. The solution proposed in this paper aims to achieve this by automating the process of collecting and labelling the training data, using it to create the model that can be used by a Deep Neural Network algorithm to accurately identify the desired individuals. A large amount of images are collected from videos using a python script and the dataset for the Deep Neural Network is collected using a face detection algorithm to located the faces in the images and a face identification algorithm to differentiate between the actors. The trained model was able to successfully identify and track the desired actors but had trouble when other non-relevant actors were present, an issue we analysed and plan on tackling in the future.

Acknowledgements

I would like to thank my supervisor Dr. Yoshi Gotoh, for his help and guidance throughout my dissertation project. I would also like to thank my parents, brother and family, whose support has been invaluable these past few years. I would like to thank my friends who have stuck with my through thick and thin and helped me relax and enjoy my time at university. Finally, I would like to express my sincere gratitude to the creators and producers of the television series Breaking Bad, from which I have used a number of shots in this dissertation. These shots have allowed me to illustrate some of the results and arguments presented in this work. However, it is important to note that these shots are not mine, and I claim no ownership or copyright over them. All rights are reserved by Sony Pictures Television and have been used based on the principles of fair use, with the proper attribution and within the limits of academic research.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Background

Using computer vision to detect and classify objects in an image or a video stream is more popular than ever, with numerous companies and organizations creating their own models to satisfy their needs. Some applications of object detection algorithms include medical imaging, object identification in satellite images, security surveillance videos, and self-driving cars.

Although extensive research is being continuously carried out in the field of computer vision with new and improved algorithms being constantly developed, creating an object detection model continues to be an extremely challenging, labour intensive and time consuming task. This project will focus on minimizing manual labor needed to train an object detection model for the task of identifying people.

1.2 Aims

The primary aim of this project is to create an algorithm that can successfully train an object detection model to identify specified individuals in an image or throughout a video stream involving multiple continuous shots while simultaneously accommodating for changes such as varying camera orientations, illuminations and weather conditions with as little human input as possible.

1.3 Objectives

The first objective is to improve the overall process of training an artificial intelligence model to identify humans by minimising manual effort. Collecting and labelling the data needed for a model is an extremely time consuming, mundane and labor intensive task that we hope to automate based on the fact that algorithms exist that can differentiate people based on their facial characteristic. Furthermore, more training data will be collected and processed in a smaller time frame allowing for increased scalability and subsequently higher accuracy.

The second objective is to design the algorithm in a way that the trained model will able to detect multiple facial features to identify the desired individuals. Since the algorithm will be primarily used on movies and TV series where actors are to be located and tracked, it is desirable for the algorithm to be able to accommodate for changes between shots where part of the actor's facial appearance may have changed (i.e. haircut, hat, glasses etc.).

The third objective is to analyse the feasibility and viability of such an algorithm. Creating any classification algorithm, especially for videos, is not a task that can be easily overcome due to the high computational requirements and massive amounts of data and time needed to process and train a model. Furthermore, a vast amount of images will not be readily available so the algorithm would have to be able to collect huge amounts of data, from different angles and under varying conditions, in a reasonable time frame. The objective aims to determine whether such an algorithm is worth implementing and whether the benefits outweigh the challenges and potential limitations.

The final objective is to ensure that the model trained using the designed algorithm can maintain high accuracy despite the type of scene depicted. The model will be applied to various samples from movies and TV series with increasing level of length and difficulty, starting with more basic scenes such as close up shots that only contain the individuals we wish to track, to more challenging shots that include crowded backgrounds and numerous individuals that we are not interested in.

1.4 Overview of the Report

Following the project introduction is the Literature Survey which will feature some of the existing research done in the specific area of computer vision that the project is targeting and how it can benefit from them. After that is the Requirements and Analysis section of the project, where the requirements of the project will be analysed along with possible methods to fulfill them. This section will also include details related to how the project will be tested and evaluated, ending with possible legal or ethical issues arising during the course of the project's development. Following that is the Design section which will give an overview of the different parts of the algorithm that will be implemented to achieve the desired result. The Implementation and Testing part of the report will be giving a detailed analysis of how the project progressed and evolved over time compared to the original design, as well as all the milestones initially set out and achieved. The Results and Discussion chapters will detail the tests conducted on the algorithm along with their corresponding results and some speculations as to why a part of the algorithm may not be performing as intended. It will also feature the effectiveness of the developed project, its feasibility, personal thoughts and some thoughts on what could have been done better or may be worth implementing in the future. The Conclusions section is going to briefly summarise the report and whether or not the objectives set out have been achieved. Finally, all sources used in this report will be

written down in the Bibliography section, and any other relevant information will be featured in the Appendix at the end of the report.

Chapter 2

Literature Survey

2.1 Overview

A great deal of research is been done in the field of object detection and classification with new research being conducted on a daily basis. In this section, we will be looking through some of the existing research that may prove useful in itself or inspire some ideas to help us with our specific problem. Face detection algorithms will be explored first, followed by known face identification algorithms and finally object detection and face recognition algorithms. Similar projects and the approaches they used will also be reviewed and taken into consideration.

2.2 Face detection algorithms

2.2.1 Haar Cascade Classifiers

One of the most famous practices for detecting human features are Haar Cascade Classifiers. They were first introduced in the research paper titled "Rapid Object Detection using a Boosted Cascade of Simple Features" (Viola and Jones, 2001) and they were the classifiers used in the first real-time face detector. Although the paper is quite old, Haar cascades remain a popular object detection algorithm, especially for facial features, because they are lightweight, extremely fast allowing them to be used for real-time applications and can achieve reasonable accuracy with relatively small amounts of training data.

At first the Haar features are collected, which involves summing the pixel intensities of regions at a specific location in the window each time and then calculating the differences between the sums. Integral images are then created to speed up the algorithm by creating sub-rectangles and array references for each of them instead of computing at every pixel. Adaptive Boosting or "Adaboost" training (Freund and Schapire, 1997) is then applied to the algorithm which uses "weak classifiers" to create a "strong classifier" that it can use to detect the desired object. Finally, using the "strong classifier", objects are indicated as found (positive) or not (negative) with the algorithm then proceeding to the next region.

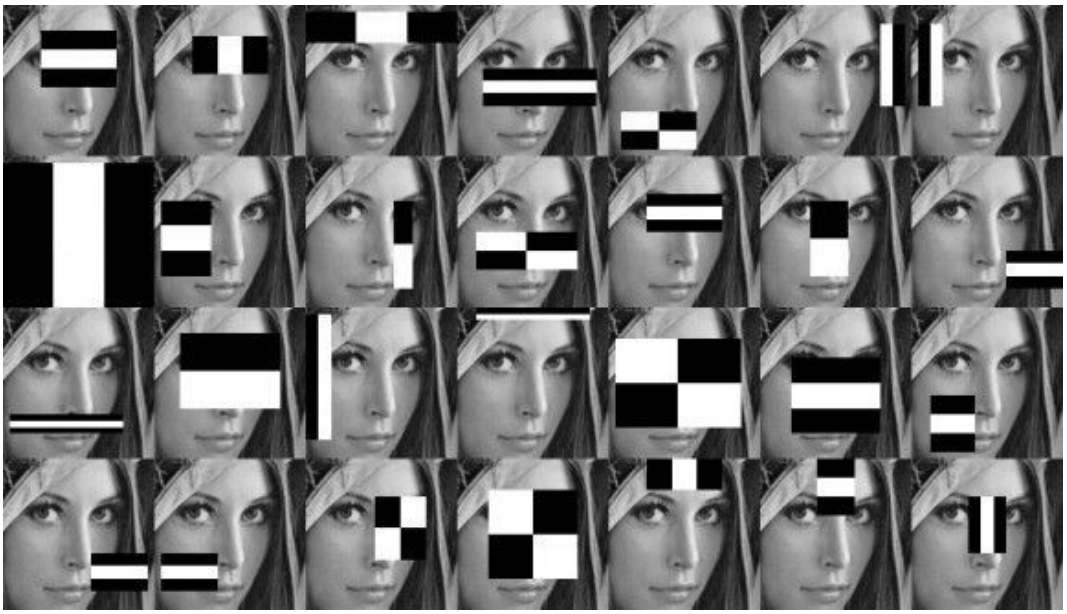

Figure 2.1: A general representation of training a Haar classifer. [1]

2.2.2 Multi-Task Cascaded Convolutional Neural Network (MTCNN)

A Multi-Task Cascaded Convolutional Neural Network or MTCNN (Zhang et al., 2016) is a deep-learning based algorithm developed by Zhang et al. in 2016 specifically designed for face detection and facial recognition. It is comprised of three stages of deep convolutional neural networks that work together to detect faces in an image.

The first stage, called Proposal Network (P-Net), proposes regions in the image that might contain a face using a lightweight CNN while the second stage, called Refine Network (R-Net), refines those regions by rejecting false positives and adjusting the bounding boxes around the faces using a more complex CNN. The third and final stage, called Output Network (O-Net), performs facial landmark detection and face classification to determine whether the detected face is a real face or a false positive. The MTCNN is known for its high accuracy and speed in detecting faces, and has been widely used in various applications such as video surveillance, facial recognition, and augmented reality. Many studies have demonstrated its effectiveness when it comes to detecting faces in challenging conditions such as low-light environments, occlusions, and different facial expressions, making it a promising approach for real-world applications of facial detection and recognition. While it shows far greater accuracy and robustness when compared to the Haar Cascades Classifiers algorithm mentioned above, the MTCNN is more computationally expensive and has a significantly longer processing time per image.

[1] Image by Edwin Jakobs (source: https://nl.pinterest.com/pin/39969515412911329/)

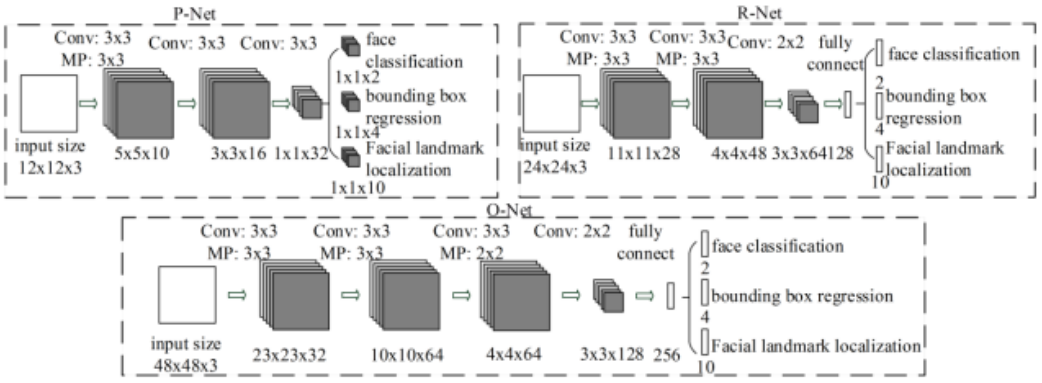

Figure 2.2: The structure of an MTCNN. (source: (Zhang et al., 2016))

2.2.3 RetinaFace

The final face detection algorithm described in this paper is RetinaFace (Deng et al., 2019). It is the most recent of the three algorithms and has gained attention due to its high accuracy, even on challenging datasets. It uses a single deep neural network to detect faces, facial landmarks, and estimate face attributes like age and gender. It also has a better performance on detecting small faces compared to the MTCNN.

RetinaFace is a face detection algorithm that uses a deep neural network to detect faces, facial landmarks, and estimate face attributes. It works by first extracting a set of feature maps from an input image using a CNN-based feature extraction network. Then, several sub-networks are applied to the feature maps to predict whether each anchor box contains a face or not, refine the location of each predicted bounding box, and estimate the coordinates of five facial landmarks for each detected face. Optionally, it can also estimate the gender and age of each detected face. The outputs of these sub-networks are combined to generate the final set of face detections, which are post-processed to filter out low-confidence detections and apply additional heuristics to improve the accuracy. Overall, RetinaFace is a highly accurate and efficient face detection algorithm that can detect faces at different scales and orientations, making it suitable for various real-world applications.

RetinaFace has the highest accuracy of all the face detection algorithms mentioned in this paper along with the highest computational requirements and processing time. Due to these disadvantages, the algorithm may not be suitable for this project as the extra time requirement does not justify the increased accuracy.

2.3 Traditional face detection algorithms

2.3.1 Eigenfaces

The fundamentals of the Eigenfaces algorithm were first presented by Sirovich and Kirby in their paper titled "Low-dimensional procedure for the characterization of human faces" (Sirovich and Kirby, 1987), and then later formalized by Turk and Pentland in "Face recognition using eigenfaces" (Turk and Pentland, 1991).

Initially each image is represented as a grayscale K×K bitmap of pixels. The images are then flattened into K^2-dim vectors which are then added to a matrix. Principal Component Analysis (PCA) is then applied to the matrix (Bartholomew, 2010).
For N images, the PCA algorithm works as follows:

- Compute the mean of each column in the matrix.
- Subtract the mean from each column (also called mean centering the data).
- Compute covariance matrix from the mean centered matrix.
- Perform an eigenvalue decomposition on the covariance matrix to get the eigenvalues and eigenvectors.
- Sort the eigenvectors in descending order.
- Select the top N eigenvectors that correspond to eigenvalues of the greatest magnitude.
- Transform the input data by projecting the dot product onto the space created by the top N eigenvectors — these eigenvectors are called eigenfaces.

To identify faces, the new image is represented by taking the dot product between the flattened input image and the N eigenfaces. The face is then classified as the face of the person whose eigenface representation has the smallest euclidean distance from the result of the previous equation. The algorithm can be improved by using more advanced machine learning algorithms, such as Support Vector Machines (Evgeniou and Pontil, 2001).

2.3.2 Local Binary Pattern Histograms (LBPH)

Proposed in 1994 (Ojala et al., 1994) and implemented in 1996 (Ojala et al., 1996), Local Binary Pattern Histograms (LBPH) are the second oldest face recognition algorithm we will explore in this paper. Local Binary Pattern (LBP) is a straightforward yet highly effective texture operator that identifies each pixel in an image by thresholding its immediate surroundings and treating the result as a binary number.

The LBPH uses four parameters.

- **Radius**: The radius, indicating the area around the central pixel, which is utilised to construct the circular local binary pattern. Typically set to 1.
- **Neighbours**: The quantity of sample points needed to create the local binary pattern in a circle. Computational cost increases as the number of sample points increases. It is usually set to 8.
- **Grid X**: The amount of horizontally oriented cells. The resulting feature vector has a higher dimensionality the more cells there are and the finer the grid is. Usually set to 8.
- **Grid Y**: The amount of vertically oriented cells. Similarly to the Grid X parameter, the resulting feature vector has a higher dimensionality the more cells there are and the finer the grid is. Default value is set to 8.

At first an intermediate greyscale image that highlights the facial characteristics is created using the radius and neighbours parameters. Each part of the image can be represented as a 3x3 matrix containing the intensity of each pixel. For each neighbour of the central value of the matrix, a new binary value is set. The value is set to 1 for values equal or higher than the central value and 0 for values lower it. The binary value gathered from the neighbours are then converted to a decimal value and set as the central value of the matrix, which is actually a pixel from the original image. Circular LBP is a variation of the algorithm that uses bilinear interpolation (Toudjeu and Tapamo, 2019). Using the image generated in the last step, the Grid X and Grid Y parameters are used to divide the image into multiple grids. Each histogram of each grid contains 256 positions representing the occurrences of each pixel intensity. Then, all histograms are concatenated to create a new, bigger one. Each class in the training set has already undergone this process during the training phase and has been assigned a histogram which is the mean of histograms from every image of that class. To assign a class to the new image, the histogram generated is compared to the histogram of each class with methods such as euclidean distance, chi-square and absolute value. The class assigned is the one with the lower distance.

2.3.3 Fisherfaces

Introduced in 1997, the Fisherfaces (Belhumeur et al., 1997) algorithm is considered an improvement over the Eigenfaces method of classifying faces by not considering illumination as a feature that represents a face. It achieves this by using Linear Discriminant Analysis (LDA) (Tharwat et al., 2017) to create the fisherfaces, an idea initially suggested by R.A. Fisher in 1936, instead of Principal Component Analysis (PCA) which is used by the Eigenfaces algorithm.

LDA is a supervised classification algorithm while PCA is an unsupervised classification algorithm. LDA discovers paths of maximum class separability while PCA discovers directions of maximum variance irrespective of class labels. While LDA functions similarly to PCA, it tries to minimise variation within each class and maximise the distance between the means of the classes. The mean of the class is its fisherface. Hence, a given image is classified as the class with the most similar fisherface, calculated using euclidean distance.

2.3.4 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (LeCun et al., 1989) are a type of artificial neural network used primarily for image recognition and processing due to their excellent pattern recognition ability. Commonly referred to as ConvNets or CNNs, they were first introduced in the 1980s by the French postdoctoral researcher Yann LeCun, who built upon the work of Japanese researcher Kunihiko Fukushima (Fukushima and Miyake, 1982). Although they were adopted early by the postal and banking sector for reading zip codes on envelopes and digits on checks, they remained largely on the sidelines due to one major flaw. They were not scalable, thus requiring a lot of data and computational resources to work efficiently for larger images. It was not until 2012, that researchers would revisit multilayered neural networks due to the large sets of data and vast computational resources that were made available.

A CNN is composed of multiple layers of artificial neurons, each neuron being a mathematical function that calculates the weighted sum of multiple inputs and then outputs an activation value based on its weights. When fed with the pixel values, the artificial neurons pick out various visual features. When an image is supplied to a CNN, each of its layers generates several activation maps which highlight the relevant features of the image. Each neuron receives a patch of pixels as input, multiplies the colours by its weights, adds the results, and then sends those results to the activation function. The first layer of the CNN usually detects basic features such as edges and its output is fed to the next layer, which extracts more complex features, such as corners. As you delve deeper into the architecture of the CNN, the layers start detecting higher-level features such as objects and faces. The final layer of a CNN is a classification layer, which takes the output of the final convolution layer as input and based on its activation map, it outputs a set of confidence scores specifying how likely the image is to belong to a class.

One of the biggest downsides of CNNs is the training required. Initially, the CNN starts off with random weights and a large dataset of images annotated with their corresponding classes is provided. It then processes each image with its random values and then compares its output with the image's correct label. If the network's output and the label contradict, which is likely the case at the beginning of the training process, it slightly adjusts the weights of its neurons so that the next time it sees the same image, its output will be a bit closer to the correct answer. Although large datasets are available to use, training a CNN to identify specific classes can be extremely time consuming and labor intensive.

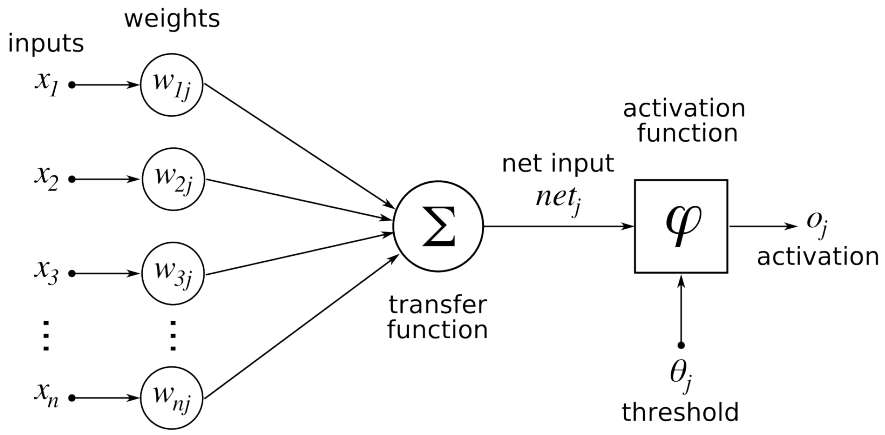

Figure 2.3: The structure of an artificial neuron (source: Wikipedia).

2.4 Two-stage object detection algorithms

2.4.1 Region Based Convolutional Neural Networks (R-CNNs)

A Region-based Convolutional Neural Network, also known as R-CNN, is a two-stage artificial neural network proposed by Ross Girshick in 2014 (Girshick et al., 2014) to improve upon the existing Convolutional Neural Networks by allowing the detection of multiple objects that appear in high frequencies. Instead of using brute force to run the CNN algorithm over every region of an image, R-CNNs extract 2000 regions from an image, referring to them as region proposals.

The 2000 region proposals are chosen using a selective search algorithm described below:

- Generate initial sub-segmentation of input image.
- Recursively combine similar bounding boxes to create larger ones.
- Use these larger boxes to generate region proposals for object detection.

The region proposals are then supplied individually to a CNN that produces the output features. These features then go through a Support Vector Machine classifier (Evgeniou and Pontil, 2001) that classifies the objects presented in each region proposal.

A basic R-CNN however is still very slow in training and testing since 2000 region proposals need to be calculated and each one needs to go through the CNN for the features to be extracted. To combat this, Ross Girshick proposed a new model called Fast R-CNN (Girshick, 2015) which was inspired by the SPPNet (He et al., 2014). In this model, the input image is supplied to the CNN rather than the region proposals, resulting in a convolutional feature map from which the region proposals are identified and warped into squares. Using a Region

of Interest pooling layer they are then reshaped into a fixed size so that they can be fed into a fully connected layer. From the Region of Interest feature vector, a Softmax layer (Bridle, 1989) is then used to predict the class of the proposed region and also the offset values for the bounding box. A Fast R-CNN is faster than a normal R-CNN because instead of feeding 2000 region proposals to a CNN one at a time, the convolution operation is done solely once per image with a feature map being generated from it.

Although Fast R-CNN is considerably faster then R-CNN, both algorithms use selective search to find the region proposals which is a very slow and time-consuming process affecting the performance of the network. An object detection algorithm developed by Shaoqing Ren replaces the selective search algorithm and allows the network to learn the region proposals itself. He named it Faster R-CNN (Ren et al., 2015). Similarly to the Fast R-CNN, the image is provided as an input to a convolutional network which in turn generates a convolutional feature map. A separate network is used to predict the region proposals instead of a selective search algorithm applied on the feature map. The predicted region proposals are then reshaped using a Region of Interest pooling layer which is used to classify the image inside of the proposed region and subsequenlty predict the offset values of the bounding boxes.

2.4.2 Region-based Fully Convolutional Networks (R-FCNs)

Introduced in 2016, Region-based Fully Convolutional Networks (Dai et al., 2016) are the newest two-stage classification algorithms, that work very similarly to R-CNNs with the main difference being that the amount of work needed for calculating each Region Of Interest is greatly reduced, making them significantly faster than the Faster R-CNN algorithm while sacrificing little accuracy.

A R-FCN divides the feature map of a class into NxN regions and creates a new feature map to detect each one of them. These maps are called position-sensitive score maps because each one detects a sub-region of the object. It then calculates the probability that each region contains the corresponding part of the object and stores it in a NxN array in a process called position-sensitive ROI-pool. The score of each class is the average of all the elements in its corresponding array. Finally a Softmax algorithm (Bridle, 1989) is applied on each score to compute the probability of each class.

In simpler terms, if C classes are supplied to be detected, a R-FCN would first expand them to C+1 so that it could include a new class for the background. N×N score maps are then constructed for each class which results in a total of (C+1)×N×N score maps. Using those score maps, a class score is predicted for each class. A Softmax algorithm is then applied on those scores to compute the probability for each class.

2.5 One-stage object detection algorithms

2.5.1 You Only Look Once (YOLO) Object Detector

One of the most prominent object detection algorithms used today is YOLO (Redmon et al., 2016). The main reason for its popularity is because the algorithm requires a single forward propagation through the neural network in order to detect objects, hence the name. As a result the algorithm is extremely fast compared to other object detection algorithms being able to output real-time, high FPS videos while at the same time maintaining high accuracy.

At first the image is divided into grids of equal size. The grids then predict the bounding box coordinates relative to their cell coordinates, the class label and the probability of the object being present in the cell. Although the computation time is greatly reduced as both detection and recognition are handled by the cells, a ton of duplicate predictions emerge. YOLO then uses Non Maximal Suppression (Neubeck and Van Gool, 2006) to suppress the bounding boxes with lower probability scores, keeping the one with the greatest probability, and then suppressing the boxes with the largest Intersection over Union with the current high probability bounding box. The process is then repeated until the final bounding boxes are obtained.

Various versions of the YOLO algorithm have been developed over the years, but we will mention the most notable ones. Among the versions created, the YOLO9000 (also known as YOLOv2) (Redmon and Farhadi, 2016) and YOLOv3 (Redmon and Farhadi, 2018) are the only ones considered "official" versions of the YOLO algorithm because they were created by some of the creators of the first algorithm, in their efforts to reduce the limitations of the original and improve its efficiency using more recent technologies that had just become available. Although the YOLOv4 (Bochkovskiy et al., 2020) and YOLOv5 versions were not created by the original authors, they are open-source projects maintained by the computer vision community and produce incredibly accurate results in extremely high speeds. Lastly, a very recent academic paper introduces the YOLOv7 algorithm (Wang et al., 2022) with initial tests showing that it surpasses all other known object detectors in both speed and accuracy, while requiring even less powerful hardware.

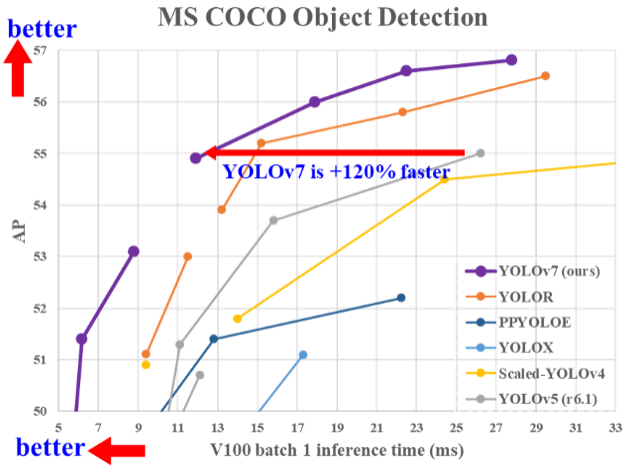

Figure 2.4: A comparison between various Yolo algorithms (source: Yolov7 repository).

2.5.2 Single Shot Detector (SSD)

Another popular single-stage object detection algorithm used today is the Single Shot Detector (Liu et al., 2015). It was designed as an improvement to the Faster R-CNN (Ren et al., 2015), which despite being state-of-the-art in terms of accuracy, is running in far below what real-time processing needs. To speed up the process, the need for the region proposal network is eliminated which causes a drop in accuracy. The SSD makes a few adjustments to offset this drop, such as multi-scale features and default boxes, which enable it to nearly match the Faster R-CNN's accuracy using lower quality images, accelerating the speed even further.

A SSD uses VGG16 (Simonyan and Zisserman, 2014) to extract the feature maps and then detects objects using the Conv4_3 layer (Yu et al., 2014). For each cell or location calculated by the Conv4_3 layer, it makes 4 object predictions. Each prediction is composed of a boundary box and 21 scores for each class, with an extra class for no object. The one with the highest score is picked as the class for the bounded object. So a total of $38 \times 38 \times 4$ predictions are created. The SSD computes both the location and class scores using small convolution filters instead of a delegated region proposal network and when the feature maps are extracted, it applies 3×3 convolution filters for each cell to make the predictions. For detecting objects independently, the SSD uses multi-scale feature maps. It uses lower resolution layers to detect larger scale objects. It also adds 6 more auxiliary convolution layers after the VGG16, with five of them being added for object detection. In three of those layers, 6 predictions are made instead of 4 which results in 8732 predictions using 6 layers. The default boundary boxes are chosen manually. The SSD initially defines a scale value for each feature map layer. Starting from the left, Conv4_3 detects objects at the smallest scale of 0.2, linearly increasing to the rightmost layer at a scale of 0.9. Combining the scale value with the target aspect ratios, the width and the height of the default boxes is computed. If the corresponding default boundary

box has an IoU (Intersection over Union) greater than 0.5 with the ground truth, the match is positive, otherwise it is negative.

To recap, the main advantage of an SSD is that it makes more predictions and has better coverage on location, scale, and aspect ratios. With the above improvements, it can lower the input image resolution to as low as 300×300 with a comparative accuracy performance. By removing the delegated region proposal and using lower resolution images, the model can run at real-time speed with accuracy almost on par with a Faster R-CNN.

In 2017, a research paper was published with the aim to improve the SSD algorithm by combining a Residual-101 (He et al., 2015) classifier with a SSD framework. The resulting algorithm was named DSSD, or Deconvolutional Single Shot Detector (Fu et al., 2017).

2.5.3 RetinaNet

The final single-stage detector described in this paper, and the one most recently developed, is the RetinaNet (Lin et al., 2017) detector. It is composed of a backbone network and two task-specific subnetworks. The backbone, which is an off-the-shelf convolutional network, computes a convolutional feature map over the whole input image. The first subnet performs convolutional object classification on the backbone's output while the second one performs convolutional bounding box regression.

To create the backbone, a Feature Pyramid Network (FPN) (Lin et al., 2016) is used on top of the ResNet architecture (He et al., 2015). While many design choices are not crucial, some modest changes are made to the FPN. The anchors used are translation-invariant anchor boxes, similar to those used in the RPN variant (Lin et al., 2016). Both subnets, work using different Fully Convolutional Networks (Long et al., 2014) that are attached to each FPN level. After thresholding the detector with a confidence score of 0.05, the network only decodes box predictions from a maximum of one thousand top-scoring predictions per FPN level. The top predictions from all levels are merged and Non Maximum Suppression (Neubeck and Van Gool, 2006) with a threshold of 0.5 is applied to yield the final detections. Thus, during training, the total focal loss of an image is computed as the sum of the focal loss over all one hundred thousand anchors, normalized by the number of anchors assigned to a ground-truth box.

The RetinaNet has extremely high accuracy, outperforming most one-stage and two-stage algorithms, while simultaneously providing a high frame rate, beaten only by the YOLO algorithm. Unfortunately, due to the use of the focal loss function in the training phase, the RetinaNet has a much higher training time then the other algorithms on the list.

2.6 Face recognision algorithms

2.6.1 FaceNet

Several deep learning algorithms have been created with the purpose of recognising human faces. One of the most notable ones is FaceNet, proposed by Google researchers in 2015 (Schroff et al., 2015). The idea behind it was to develop a high-dimensional feature space for faces, where the distances between faces represent how similar they are to one another.

FaceNet works by taking as input an image of a face and producing a feature vector with a size of 128, referred to as embedding, as output. The embedding is generated by passing the input image through a deep neural network that is trained to map faces to a feature space. The neural network used is based on the Inception architecture. However, instead of classifying images into a set of predefined classes, the network is trained to output a continuous vector representation of the face. The training process involves using a large dataset of face images, where each face is labeled with an identity. The model is trained using a triplet loss function, which encourages the model to produce embeddings that are closer together for faces of the same individual and further apart for faces of different individuals. To achieve this, the training process involves selecting three images for each training sample: an anchor image, a positive image (a different image of the same individual as the anchor), and a negative image (an image of a different individual). The model is then trained to minimize the distance between the anchor and positive embeddings, while maximizing the distance between the anchor and negative embeddings. The resulting embeddings can then be used for a variety of tasks, including face recognition, face verification, and clustering. To recognize a new face, the input image is passed through the trained model to obtain its embedding, and the distance between this embedding and the embeddings of previously seen faces is computed to determine the closest match.

FaceNet is an extremely accurate and robust algorithm, able to recognise a face correctly with a probability of up to 99.63% on certain datasets. However, it requires huge amounts of data and significant computational resources to operate accurately. Furthermore it was primarily designed to recognise faces in images instead of videos, for which it does not perform as well. Its speed declines significantly and it does not adapt well to the changes such as lighting. These limitations do not diminish the performance of the FaceNet algorithm however they do suggest that it may not be the ideal solution to the problem being tackled in this paper.

2.6.2 DeepFace

Another famous face recognition algorithm is DeepFace developed by the Facebook AI research team in 2014 (Taigman et al., 2014). It was trained on four million photographs shared by Facebook users and employs a nine-layer neural network with approximately 120 million connection weights.

The input image is initially preprocessed to make sure that all faces are aligned and normalized to a standard size and orientation. This is done to reduce variations in lighting, pose, and facial expressions that can affect the accuracy of facial recognition. DeepFace uses a 3D model to detect facial landmarks and applies affine transformations to align and normalize the face. It then uses a CNN to extract features from the face. The CNN architecture is a deep network that has many layers of convolutional and pooling operations, followed by fully connected layers. The network has been trained on a large dataset of faces to learn representations that are useful for face recognition. The CNN outputs a high-dimensional feature vector that represents the input face. Once the features have been extracted, the algorithm calculates the similarity between the input face and a set of known faces in its database. This is done using a metric learning algorithm called Siamese network (Bromley et al., 1993), which learns a distance function that measures the similarity between two faces. The Siamese network consists of two identical CNNs that share the same weights. The input faces are fed into the two CNNs, and the output feature vectors are compared using a distance metric, such as the cosine similarity or Euclidean distance. The distance metric outputs a similarity score that indicates how similar the two faces are. Finally, the system classifies the input face by matching it with the known face that has the highest similarity score. If the similarity score exceeds a certain threshold, the system will recognize the face as a match. The database of known faces was originally populated by collecting face images from social media and tagging them with names. DeepFace uses a clustering algorithm to group similar faces together and automatically creates name tags for each cluster.

DeepFace was designed primarily for face verification and identification in social media settings, where users need to be matched to their photos. Similarly to the FaceNet algorithm, DeepFace was designed to recognise faces in still images and not videos, especially movies or series. A few approaches exist that may be combined with FaceNet to increase its performance on videos, however they are computationally expensive and require significant processing power, making them impractical for processing videos of great length.

2.7 Relevant work

2.7.1 Face Recognition: Traditional versus Deep Learning methods

In a paper published by the Madhav Institute of Technology and Science in Gwalior, India, (Jayaswal and Dixit, 2020), Ruchi Jayaswal and Mansih Dixit attempted to compare traditional and deep learning methods in terms of accuracy when it come to detecting and recognising faces in a real-time environment. A dataset consisting of 300 images, 100 images per person, was collected and used to train both methods. The traditional face detection method used Haar cascades to detect faces in the images and the LBPH algorithm to classify them, resulting in an accuracy of 50%. The deep learning approach extracted the faces using the MTCNN algorithm (Zhang et al., 2016), with the FaceNet model (Schroff et al., 2015) being used to extract the highest quality features from images and categorize each face. This

method yielded an accuracy of 92% on the dataset. The results of the tests conducted led the researchers to the conclusion that the traditional face identification methods were viable for small datasets while the deep learning approach was better suited for larger datasets.

2.7.2 Student attendance with face recognition (LBPH or CNN)

Researchers at Bina Nusantara University (Budiman et al., 2023) tried to compare the traditional LBPH face recognition algorithm to a CNN when it comes to monitoring attendance at universities using a Systematic Literature Review. The researcher identified several studies that demonstrated the accuracy and efficiency of these systems, with some achieving recognition rates of over 95%. Many of them highlighted the potential benefits of using face recognition technology in educational settings, including improved efficiency, reduced administrative workload, and increased security. From a review done on 30 such articles, it was concluded that a CNN would be more suitable for a class attendance system due to its high accuracy and stability under the influence of external factors. They concluded that the fundamental disadvantage of utilising a CNN algorithm compared to the LBPH algorithm is the necessity for many datasets, hence an effective method of gathering them is essential. Regardless of the approach chosen, model performance was heavily impacted by outside variables including backdrop, illumination, and face position. They argued that further developments are still possible, such as selecting a face detection technique that is compatible with the current face recognition system and that the system's accuracy can be improved further by using face detection techniques that complement face recognition in order to provide optimal accuracy.

2.7.3 Identifying actors in movies

Despite facial recognition technology's enormous achievements, it is still difficult to identify people in natural circumstances. Issues with profile views, poor lighting, and obstructions might cause serious problems. Early techniques generally attempted to leverage more visual clues such clothes or more metadata (Sumengen et al., 2007). Nonetheless, the progress was quite modest. Subsequently, more complex frameworks that incorporate several cues such as clothes, timestamps and sceneries were created (Lin et al., 2010). The Chinese University of Hong Kong recently attempted to combine visual cues and instance-dependent weights to identify actors in films, surpassing all previous techniques. (Huang et al., 2018).

Chapter 3

Requirements and Analysis

3.1 Overview

This chapter will cover the general project requirements along with the features that the chosen solution should include to be considered successful. Additionally, it will go over various existing algorithms explored in the Literature Survey chapter and highlight the ones most likely to allow us to achieve the desired result. Following that, any ethical, professional and legal issues raised by the project will be discussed.

3.2 Project Requirements

The requirements of the project are the minimum of what the implementation of the algorithm must be able to perform in order to meet its aims and objectives.

- **Accurate**. The resulting model must be able to accurately identify the specified individuals and distinguish them from all others regardless of external parameters such as lighting, weather conditions, etc.
- **Efficient**. It must be able to process the given work load and return the desired result in a reasonable amount of time.
- **Adaptable**. The project should be able provide a solution for all given types of input and be able to support any changes in video recording or object classification techniques with as little modification as possible.
- **Reasonable model training time and cost**. For the project to be useful in a professional environment, it must be able to keep up with the nature of the film/series industry. Actors clothes as well as their characteristics (i.e facial hair, hair color, hair style) may change from scene to scene or episode to episode. Thus, adding more samples to the model and retraining it must be a relatively quick and easy process.
- **Minimal user input**. The algorithm should be able achieve its goals with as little manual effort as possible and using just the initial inputs provided by the user.

3.3 Possible techniques and tools

3.3.1 Programming language and libraries

The programming language that will be used for creating the program is **Python 3.9.7**. Some open-source libraries that will most likely be used include:

- **OpenCV**. Provides programming functions mainly aimed at real-time computer vision.
- **Pillow**. Adds support for opening, manipulating, and saving different image file formats.
- **Numpy**. Adds support for large, multi-dimensional arrays and matrices along with a large collection of high-level mathematical functions for their manipulation.
- **Pandas**. Offers data structures and operations for manipulating numerical tables and time series.
- **TensorFlow**. Focuses on training and inference of deep neural networks.
- **Keras**. Acts as an interface for the TensorFlow library.
- **PyTorch**. A machine learning framework based on the Torch library and mainly used in computer vision and natural language processing applications.
- **Matplotlib and Seaborn**. Allow embedding plots into applications.
- **Albumentations**. Provides fast and flexible image augmentations (Buslaev et al., 2020).
- **Pickle**. Used for serializing and deserializing a Python object structure.

3.3.2 Possible Techniques

During the development of the project, many known computer vision algorithms and techniques may be used to determine which ones are best suited for our needs, if any, based on their accuracy and efficiency. Out of all the techniques described in the Literature Survey, the ones most likely to be implemented are:

- **Haar Cascade Classifiers**
- **MTCNN (Multi-Task Cascaded Convolutional Networks)**
- **Eigenfaces, Fisherfaces and Local Binary Pattern Histograms**
- **YOLO (You Only Look Once) Algorithm**
- **Faster R-CNN (Region-Based Convolutional Neural Network)**

The Haar Cascade Classifiers method is likely to be used because of the existing cascades supplied by the OpenCV library, nullifying the need to train the algorithm in identifying different human body parts and facial features. The Multi-Task Cascaded Convolutional Neural Network (MTCNN) on the other hand is a pre-trained neural network especially designed to detect faces and facial landmarks in images. It is more computationally expensive when compared to Haar Cascades but it has significantly higher accuracy.

Feature	**Haar Cascades**	**MTCNN**
Speed	Very fast (> 30 FPS), real-time	Fast (>10 FPS), real-time
Accuracy	Good	Very good
Robustness	Bad	Very good
Using GPU	Certain implementations (not OpenCV)	Yes
Using colour	No	Yes

Table 3.1: Comparison of Haar Casscades and MTCNN detectors

Traditional face recognition methods will be used to identify which face corresponds to each actor to collect the training data for the object detection algorithm. Out of the three algorithms described in the Literature Survey, tests will be conducted to discover the algorithm that is best suited for this particular scenario.

For the final part of the algorithm, the YOLOv7 or Faster R-CNN algorithms will most likely be used due to their easy implementation and them having the highest accuracy out of all the object detection algorithms. From those two, the YOLOv7 algorithm is more likely to be used because of the much higher processing time unless the lower accuracy when compared to a Faster R-CNN proves to be an issue.

3.4 Testing and evaluation

This project uses computer vision to track individuals in a video and as a result of this open-ended nature it will involve a number of manual experiments that are designed to test the capabilities of the algorithm under various different situations. Multiple different scenes will be used and an evaluation will be completed on how well the algorithm performs on each one.

Numerous algorithms such as Yolov7 include a built-in testing functionality where the trained model is applied on a supplied dataset and a log as well as several figures and data points assessing the efficacy of the model on the test set are saved and returned in a specified file. The total number of photos in the test dataset, the number of labels assigned to each category in those images, as well as the precision, recall, and Mean Average Precision for the cumulative predictions and each sort of classification are all displayed in the logs. Manual tests would also need to be carried out to ensure that the accuracy is not affected by the video or image quality, format, etc.

The efficiency of the algorithm will be measured by the length of time needed to process a video from start to finish while taking into account the number of individuals tracked. The training time of the algorithm will be highly dependent on the size of the training data. Tracking more individuals will probably result in longer training times which may be a deciding factor for the overall feasibility of the algorithm.

3.5 Ethical, Professional and Legal Issues

3.5.1 Malicious use of tracking software

Due to the nature of the project, the possibility of the program being used for unlawful or malicious purposes must be considered. It has been argued that tracking software is against Article 8 of the European Convention on Human Rights, the right to a private and family life however no specific facial recognition law exists in the UK. Although the actions of other people can not be controlled from our end, the intended use of the program should be clearly stated.

3.5.2 Permission to use footage from movies/series

According to section 29 in the Copyright Exception part of the Copyright, Designs and Patents Act of 1988, a person is allowed to copy limited extracts of works when the use is non-commercial research or private study in order to allow students and researchers to make limited copies of all types of copyright works for their own research purposes. To use a larger amount of extracts the work itself must be free-to-use or permission must be granted by the rightful owner of the material.

Chapter 4

Design

4.1 Overview

This chapter focuses on the proposed design of the algorithm and the reasoning behind the choices made. The solution's design needs to be capable of meeting the majority, ideally all, of the original specifications listed in the preceding chapter while still being of reasonable scope and complexity.

4.2 Algorithm Design

4.2.1 Initial algorithm design

The initial design of the algorithm is illustrated in the flow chart below.

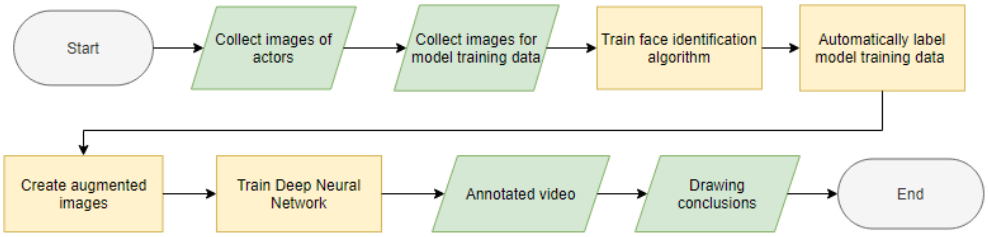

Figure 4.1: Flowchart of the proposed algorithm.

Each stage of the algorithm will be described below along with the reasoning behind all of the choices made.

4.2.2 Data collection

The training data used for the creation of the model will be composed of images from two different sources.

The first source is videos with the images being extracted using a python script. Initially, a specified time frame and interval will be requested. The script will then automatically scan the video and during the specified time frame it will store the frame between every given interval. For example, if the interval is set to 15 frames, every 16th frame will be saved. The purpose of the first part is to collect images of the actors as they will appear in future videos that will be analysed. This part can be repeated any number of times for different videos, without deleting the previous data.

The second source is public google images. All images obtained from the internet must carry a creative commons license or permission must be granted by the owner of the image. The motivation behind this step is to add images of the actor throughout different stages of their life and introduce variety to the dataset. This portion of the data collection can be done once and does not have to be repeated.

Finally, once the previous images of the actors have been collected and processed using the face identification algorithm, the Albumentations library will be used to create additional images by applying visual effects to the existing ones such as changing the orientation, adding movement blur or changing the weather conditions (i.e. introducing fog or rain). The idea behind this final part is to expand the dataset with images that are not readily available.

4.2.3 Detecting faces in images

Manually drawing bounding boxes and labeling them in each individual image, for hundreds or thousands of images, is an extremely time-consuming and mundane process which we are aiming to avoid by training an algorithm to do it for us. Before the face identification algorithm can be implemented to collect the training dataset from the complete set of images collected, it has to be trained to detect the desired actors. Haar Cascades or the MTCNN will be used to automate the process of detecting the faces of the actors. Fortunately, both of these algorithms are provided from python libraries making the process of implementing them quicker and more straight-forward.

4.2.4 Training the face recognition algorithm

The traditional face classification algorithms will be trained to automatically match the faces detected to their corresponding class. To automate this process it is required for the images to only contain one face per image as to not require the approval of a human operator and speed up the process. This however means that the operator has to check each image supplied otherwise the performance of the algorithm may drop due to incorrect data.

The time required to perform the training will grow based on the number of actor we wish to track, the face detection algorithm we use, and the number of images per actor. For example, the MTCNN algorithm is more time-consuming than using Haar Cascades to detect faces but has far superior accuracy directly affecting the quality of the algorithm. A traditional face

detection algorithm does not require nearly as many images as a CNN with the optimal number of images being about 100 per class according to various sources. It is important to have roughly the same amount of images per class so that the algorithm does not develop a bias towards the class with the most images.

4.2.5 Training the Deep Neural Network

Obtaining the training data for the Deep Neural Network will be by far the hardest and most important task that we will face. Due to the incredibly high volume of labelled images needed for this portion of the algorithm, it is our intend to optimise the process using the traditional face classification algorithm trained in the previous step.

Initially, the algorithm will read images from a folder and detect the faces in the images using Haar Cascades or a MTCNN similarly to how it will be implemented in the previous phase. In this phase however, the time difference between the two approaches is sure to be far more noticeable due to the input data being several times larger than previously. It will then predict which actor that face belongs to by calculating the distance between the face detected and the average of the faces per actor, keeping the actor whose average results in the less distance from the detected face.

Finally, after all of the images are labelled, they will be processed using the Albumentations library to generate augmented images from the originals, along with new corresponding modified labels.

4.2.6 Identifying and classifying individuals in a video

The next task is the identification and classification of humans in the video stream. The two algorithms most suitable for this task are the YOLOv7 and Faster R-CNN algorithms due to the high accuracy they offer. Out of these two algorithms YOLOv7 has a much lower processing time which is highly desirable when it comes to processing videos of great length such as movies or TV episodes. It is worth noting here that the Deep Neural Network used in this stage should be interchangeable and the algorithm should not depend on a specific one to operate correctly. It will be highly beneficial if the algorithm is able to accommodate for any and all changes with as little modification as possible.

Using the model trained for the Deep Neural Network, the algorithm should now be able to detect and identify the desired actors and create bounding boxes around them with the label attached. If multiple overlapping bounding boxes with the same label are drawn, a new bounding box which includes all of them will be drawn. This is achieved using Non Maxima Suppression and Intersection over Union, both of which are already included in the base Yolov7 version.

4.2.7 Drawing conclusions from the data gathered

Through the previous stages data such as the number of frames in which each actor is present will be collected. The data can then be used to draw conclusions about the video and its contents. For example, the screen-time of each actor can be calculated and the actors can be assigned roles based on those calculations (i.e. protagonist, deuteragonist etc.).

Chapter 5

Implementation and Testing

5.1 Overview

The implementation section will provide a detailed account of the methods and procedures that were used to carry out the project. It includes the detailed implementation process for each part of the algorithm that was described in the previous chapter. Additionally, this section will address any limitations or challenges encountered during the implementation process and discuss how they were addressed.

5.2 Collecting the training data

5.2.1 Collecting images from the video stream

As specified in the previous sections, the training data will include images obtained from videos which included the actor(s). To achieve this, a python script has been created which takes five parameters as input.

- The location of the video file.
- The name of the video file.
- Directory to store the resulting images.
- The frame interval.
- The time frame(s).

The script uses the Path library to access the video from the specified directory and the OpenCV library to load it. If the directory in which the images are to be stored does not exist, it is created. Once the video has been loaded the validity of the time frame is checked and if it is valid, the video jumps to that frame. While inside a given time frame, each frame between the specified interval is resized and stored in the target directory with the name of the video and the number of the frame. The process ends once the end of the time frame is

reached. If multiple time frames have been specified, the process repeats for each one. If the script has traversed all given time frames, the program terminates. The initial design of the script would only accept one time frame at a time but was later modified as to avoid wasting time by requiring the user to re-run the script multiple times for the same video.

```
# Check validity of time frame
def check_valid(start_frame, end_fame, total_frames):
  if start_frame > end_frame:
        print("Start frame is greater than end frame. Moving to next set.")
        return False

  if start_frame > total_frames:
        print("Start frame is greater than total frames. Moving to next set.")
        return False

  return True


for i in range(len(minutes)):

    minute_tuple = minutes[i]
    start_frame = int((math.floor(minute_tuple[0]) * 60 + ((minute_tuple[0] -
        math.floor(minute_tuple[0])) * 100)) * fps)
    end_frame = int((math.floor(minute_tuple[1]) * 60 + ((minute_tuple[1] -
        math.floor(minute_tuple[1])) * 100)) * fps)

    if not check_valid(start_frame, end_frame, total_frames):
        continue

    cap.set(cv2.CAP_PROP_POS_FRAMES, start_frame)

    frame_id = start_frame

    while(cap.isOpened() and frame_id < end_frame):
        ret, frame = cap.read()

        if ret == True:
            if frame_id % skip_frame == 0:
                # Display the resulting frame
                if resize:
                    frame = cv2.resize(frame, resize_size)
                cv2.imshow('Frame', frame)
                cv2.waitKey(1)
                cv2.imwrite(result_folder + "/" + img_names
                    + str(frame_id)+'.jpg', frame)

        frame_id += 1
```

Listing 5.1: Frame collection from video

5.2.2 Collecting images from the web

Although a script that downloads images directly from Google can be written using the extremely popular "Requests: HTTP for Humans" library (Reitz, 2011), there is no guarantee that the images downloaded are licensed for public use under a Creative Commons license. To avoid any potential legal issues, it is necessary to manually download the required images while verifying their licensing terms.

5.3 Detecting faces

Two distinct approaches have been employed for face detection. The first one is based on Haar Cascades Classifiers and the other one utilizes a MTCNN. Each method has its own advantages and limitations and their comparative performance will be evaluated.

5.3.1 Haar Cascade Classifiers

At first, two cascade objects are created using the public cascades provided by the Opencv library. The first one detects the frontal view of a face while the second one detects the profile view of a face. For each image the cascade objects detect the faces in the image using the following parameters:

- **Image:** Parameter specifying image to scan in greyscale.
- **Scale factor:** Parameter specifying how much the image size is reduced at each image scale.
- **Minimum number of neighbors:** Parameter specifying how many neighbors each candidate rectangle should have to retain it.
- **Minimum face size:** Parameter specifying the minimum possible object size.
- **Maximum face size:** Parameter specifying the maximum possible object size.

For each face detected, the x and y coordinates of the upper left corner of the rectangle are saved in a 2-D array along with its width and height. The array has dimensions faces_detected x 4.

To achieve optimal performance, manually tuning the parameters of the classifiers is required. The minimum and maximum face size parameters do not affect performance by much but they can be used to eliminate outliers. The scale factor most commonly used lies in the 1.05∼1.50 range. A smaller scale factor results in more image scales being processed increasing detection accuracy and computation time while a larger scale factor results in fewer image scales being processed decreasing the computation time and detection accuracy. The most common number of neighbors used lies in the 2∼5 range. A higher value of neighbors results in fewer false positives but may also result in missed detections while a lower value increases the number of detections but may also increase the number of false positives.

Figure 5.1: Face detection using Haar Cascades and a combination of parameter values.

Low scale factor, low number of neighbors	Low scale factor, high number of neighbors
High scale factor, low number of neighbors	High scale factor, high number of neighbors

Table 5.1: Values of parameter for each image.

The above figure showcases the difference made by tuning the scale factor and number of neighbors of the Haar Cascade Classifier. From the tests conducted, the ideal value for the scale factor seems to be 1.5 and the number of neighbors 3.

5.3.2 MTCNN

Using the pre-trained MTCNN requires installing and importing the mtcnn python library (Paz Castro, 2016). When the library, along with all of its' dependencies are installed, the MTCNN detector is initialised. The detector is supplied a BGR version of the image (instead of the standard RGB format) and returns the following values:

- **Box:** The x and y coordinates of the upper left corner of the rectangle along with its width and height.
- **Keypoints:** The x and y coordinate for the face features (nose, right mouth edge, right eye, left eye and left mouth edge).
- **Confidence:** The probability for a bounding box to be matching a face.

Figure 5.2: Face detection using the MTCNN.

```
if FACE == "mtcnn":
    detector = mtcnn.MTCNN()
else:
    face_cascade_front = cv2.CascadeClassifier(
        'cascades/data/haarcascade_frontalface_alt.xml')
    face_cascade_profile = cv2.CascadeClassifier(
        'cascades/data/haarcascade_profileface.xml')
```

Listing 5.2: Initialize face detection algorithm

5.4 Training the face recognition algorithm

Initially, the images of each actor are organized into individual sub-folders named after the respective actor. The root folder, which contains all the sub-folders, is supplied as a parameter when the script is invoked. Two folders are subsequently created, if they do not already exist, with one folder being responsible for storing a dictionary bounding each actor to a unique id using the Pickle library and the other for storing the trained model of the traditional algorithm in .yml format. The chosen face detection and face classification algorithms are also initialised.

Using a for-loop and the os python library, the files are read from the sub-folders one-by-one. When a file is read from a sub-folder, its extension is tested to ensure that the file is an image and skips anything with an extension belonging to a different kind of file. The image is then read by OpenCV, its' dimensions are saved and a greyscale version of the image is generated.

Every time an image from a new sub-folder is read, the name of the sub-folder is saved in a dictionary as the label for that class along with a unique id.

The face detection algorithm is used to locate all faces present in an image (ideally only one face is detected). The script iterates over each image, a greyscaled version if Haar Cascades are choosen and a BGR version if MTCNN is choosen. When a face is located, a region of interest is selected from the greyscaled image (since all face identification algorithms are trained on a greyscale image) and is resized if eigenfaces or fisherfaces is selected. Once all files have been processed, the dictionary housing the labels and their ids is stored in a pickle file and the face classification algorithm is trained on the data kept before the model is saved in .yml format.

```
for image in images:

        if image.endswith(('.jpg', '.png', 'jpeg', 'webp', '.JPG',
            '.PNG', 'JPEG', 'WEBP')):

            current_image += 1
            print("Processing image " + str(current_image) + " of "
                + str(number_of_images) + " in folder " + folder)

            new_image = os.path.join(subfolder_path, image)
            img = cv2.imread(new_image)
            img_height, img_width, _ = img.shape
            if img_height > 1080 or img_width > 1920:
                img = cv2.resize(img,(1920,1080))
                img_height, img_width, _ = img.shape
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

            if not label in label_ids:
                label_ids[label] = current_id
                current_id += 1
            id_ = label_ids[label]

            if FACE == "mtcnn":
                img_bgr = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
                faces = detector.detect_faces(img_bgr)
            else:
                faces = face_cascade_front.detectMultiScale(gray,
                    scaleFactor=1.5, minNeighbors=5)
                faces2 = face_cascade_profile.detectMultiScale(gray,
                    scaleFactor=1.5, minNeighbors=5)

                # Combine the two sets of faces
                if len(faces) != 0 and len(faces2) != 0:
                    faces = np.concatenate((faces, faces2))
                elif len(faces) == 0 and len(faces2) != 0:
                    faces = faces2

            if len(faces) == 0:
```

```
                print("No faces found in image: " + image)
                images_kept -= 1
                continue

            if len(faces) > 1:
                images_with_multiple_faces +=1

            for face in faces:

                if FACE == "mtcnn":
                    x, y, w, h = face['box']
                else:
                    x, y, w, h = face

                roi = gray[y:y+h, x:x+w]
                if RECOGNIZER == "eigen" or RECOGNIZER == "fisher":
                    roi = cv2.resize(roi, (48,48))
                x_train.append(roi)
                y_labels.append(id_)
```

Listing 5.3: Collecting faces from images

5.5 Face recognition algorithm comparison

The three face recognition techniques readily available by the OpenCV library are Eigenfaces, Fisherfaces and Local Binary Pattern Histograms. Existing research has been done comparing these techniques under different circumstances. Several academic papers have been published comparing the accuracy and limitations of each method for various types of applications.

A thesis conducted by the Uppsala University (Norvik, 2022) compared the three algorithms to evaluate which one was best suited for access control system for heavy vehicles. In their study, the LBPH algorithm outperformed both the Eigenfaces and Fisherfaces algorithms showing the highest precision and the least false positive detections. The thesis also compared the time taken by each algorithm and concluded that there isn't a noticeable difference between them.

Another paper by Ahsan et al. (Ahsan et al., 2021) evaluated these three algorithms with images captured under different weather conditions and subsequently on three different datasets. The paper concluded that the LBPH algorithm showed the highest accuracy overall and the second lowest execution time.

Figure 5.3: Results obtained from a small test. The order of the algorithms is Eigenvalues(Top-left), Fishervalues(Top-right), LBPH(Bottom).

The results obtained from previous work done and the similar nature of the images suggested that the traditional face classification algorithm best suited for this project is the Local Binary Pattern Histograms algorithm. However the proper accomodations have been made so that all three algorithms can be implemented and further tests will be conducted on the full set of training data to determine the viability of each one.

```
if RECOGNIZER == "lbph":
    recognizer = cv2.face.LBPHFaceRecognizer_create()
elif RECOGNIZER == "eigen":
    recognizer = cv2.face.EigenFaceRecognizer_create()
elif RECOGNIZER == "fisher":
    recognizer = cv2.face.FisherFaceRecognizer_create()
```

Listing 5.4: Initialize face identification algorithm

5.6 Automatically labeling images

Prior to running the python script, the folders containing the images to be used for training, which were collected in section 5.2, are moved inside of a new folder. At first the root folder where all of the images were moved is specified and the output paths are created if they do not already exist. Training an object detection model requires a train folder, a validation folder, and a test folder. Inside of each folder two sub-folders are created one storing the images and the other their corresponding labels. Subsequently, the chosen face recognition and face classification algorithms are initialised as before. The model trained in the previous

section is then loaded in the classification algorithm and the pickle file is unwrapped in a new dictionary.

```
if RECOGNIZER == "lbph":
    recognizer = cv2.face.LBPHFaceRecognizer_create()
elif RECOGNIZER == "eigen":
    recognizer = cv2.face.EigenFaceRecognizer_create()
elif RECOGNIZER == "fisher":
    recognizer = cv2.face.FisherFaceRecognizer_create()

recognizer.read("recognizers/face-trainner.yml")

labels = {"person_name": 1}
with open("pickles/face-labels.pickle", 'rb') as f:
    og_labels = pickle.load(f)
    labels = {v:k for k,v in og_labels.items()}

if FACE == "mtcnn":
    detector = mtcnn.MTCNN()
else:
    face_cascade = cv2.CascadeClassifier(
        'cascades/data/haarcascade_frontalface_alt.xml')
    face_cascade2 = cv2.CascadeClassifier(
        'cascades/data/haarcascade_profileface.xml')
```

Listing 5.5: Initialize face detection and identification algorithms

Using the os python library and a for-loop, each sub-folder is navigated. The number of files in the sub-folder are recorded and thresholds for training, validation and test sets are created. The recommended distribution is 70% training data, 20% validation data and 10% test data. A counter is also initialised to keep track of where each image belongs, or it is reset if it already exists. For each file inside the sub-folder, the file's extension is checked to ensure that it is an image, skipping all files without an appropriate extension. The image is then copied to the image sub-folder of the appropriate folder, depending on what the threshold counter is, and a text file is created in the labels sub-folder of the same folder with the same name as the image. Once again, the image is read by OpenCV, its' dimensions are saved and a greyscale version of the image is generated.

The faces in the image are identified using the selected algorithm and if no faces are detected both the image and label files are deleted and the program moves on to the next image. For each face found, the coordinates of the top left corner of rectangle surrounding the face along with its' width and height are saved. The area of the grayscale version of the image contained in the rectangle is supplied to the recognizer which calculates the distance between the face detected and the average of all the faces for each class returning the id and distance of the class whose mean is the closest to the image. To ensure that the face belongs to the correct actor, any face whose confidence is higher than a specified threshold value (distance

between face and class) is discarded. If the confidence of the face is within the accepted limit, a new line is added to the text file containing the label for that face. Not all object detection algorithms use the same labelling format. The Yolo algorithm uses the following format: [label id, normalised x center coordinate, normalised y center coordinate, normalised width, normalised height]. An R-CNN on the other hand uses the standard pascal_voc format: [label id, x coordinate of the top left corner, y coordinate of the top left corner, x coordinate of the bottom right corner, y coordinate of the bottom right corner]. The process repeats for all the faces in the image. If no faces have a confidence less than the threshold, both the image and text file are once again deleted to reduce storage capacity. The script terminates when all images are processed.

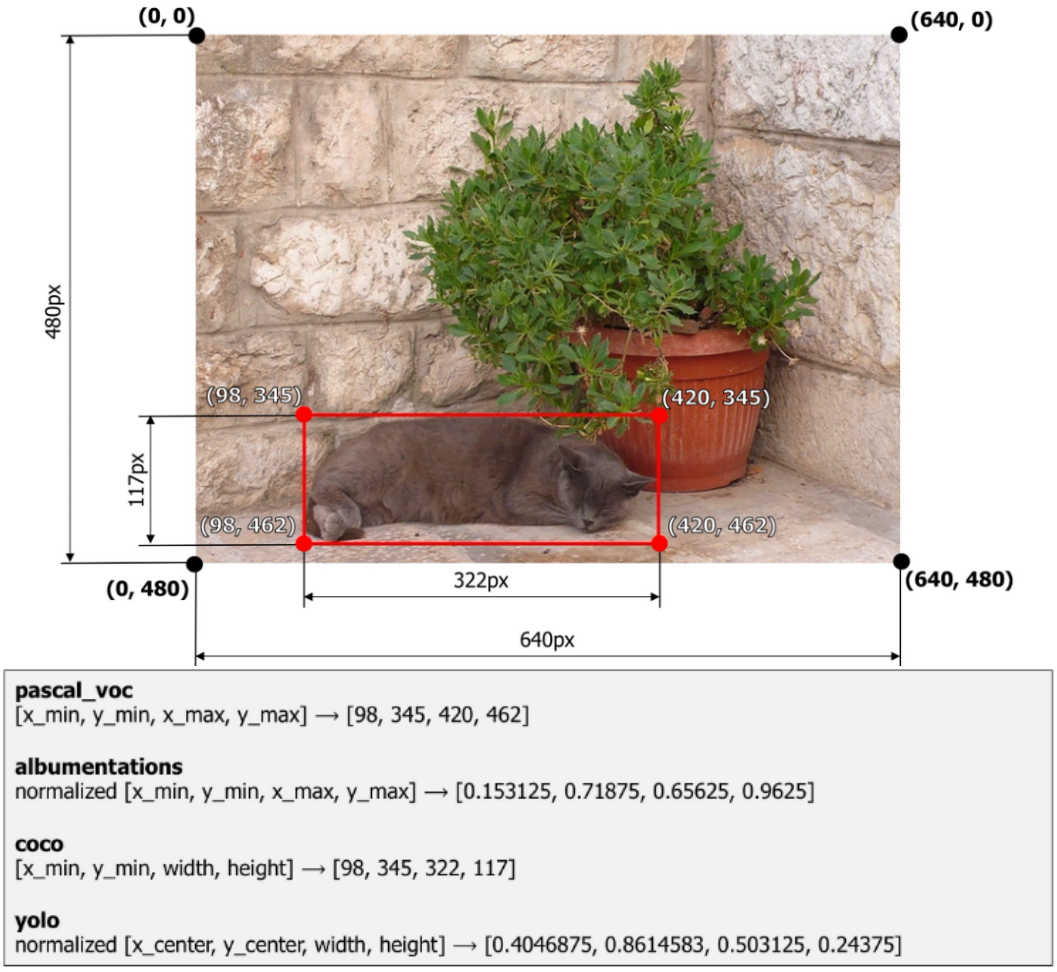

Figure 5.4: Different labelling formats (source: Albumentaions library documentation).

5.7 Augmenting images in the training set

Once all images have been labelled and moved to their respective folders, the Albumentations library is used to expand the dataset and generate images with visual effects, not easily obtained. Nearly all papers that describe state-of-the-art image recognition algorithms include basic augmentation techniques. In a paper published in 2018 from Google about AutoAugment (Cubuk et al., 2018), an algorithm that automatically discovers the best set of augmentations for a dataset, showed that a custom set of augmentations improves the overall performance of a model.

The script receives the path of the images, the path of the labels for those images and the number of images to generate. The script initially adds all the images from the given directory to a list. It then chooses a random image from the list and reads it using OpenCV before removing it from the list.

```
# p is the probability of applying the augmentation in the image
transform = A.Compose([
    A.ShiftScaleRotate(shift_limit = 0.08, scale_limit = 1.4,
    rotate_limit = 45,
    interpolation = 1, border_mode = 0, p = 0.4),
    A.OneOf([
        A.MultiplicativeNoise(p = 0.5),
        A.GaussNoise(p = 0.5),
    ], p = 0.2),
    A.OneOf([
        A.MotionBlur(p = 0.4),
        A.MedianBlur(blur_limit = 3, p = 0.2),
        A.Blur(blur_limit = 3, p = 0.2),
    ], p = 0.3),
    A.OneOf([
        A.Sharpen(),
        A.Emboss(),
    ], p = 0.5),
    A.CLAHE(clip_limit = 2),
    A.RandomContrast(limit = 0.2),
    A.OneOf([
        A.RandomRain(p = 0.2),
        A.RandomSunFlare(p=0.2),
        A.RandomFog(p=0.2),
        A.RandomSnow(p=0.2),
        A.RandomShadow(p=0.2),
        A.RandomBrightness(p=0.2),
    ], p = 0.5),
    ],
    bbox_params = A.BboxParams(format = 'yolo',
        label_fields = ['category_ids']))
```

Listing 5.6: Augmentaton probabilities

The above code snippet outlines the process of applying probabilistic image augmentation using the Albumentation library. The probability of each effect being applied to the input image is specified through the 'p' parameter, followed by the definition of the output labeling format on line 30. The script then proceeds to read each line of the corresponding label file and stores the object IDs and box coordinates in separate lists. Subsequently, the augmented image is created using the defined probabilities and a new bounding box surrounding each face is generated in the specified labeling format. The resulting image is saved under a new name in the images directory along with the corresponding label file which is also saved with the same name. The script terminates once the desired number of augmented images has been generated.

Figure 5.5: Example of an augmented image

5.8 Training the object detection algorithm

Training the Yolov7 algorithm is a fairly straight forward process. The train, validation and test folders need to be moved inside of the data directory and a new data file has to be created in .yml format which defines the paths to the three folders, the number of classes to detect, and the names of the classes. It is pivotal to list the names of the classes in the same order as they were listed when the face classification algorithm was trained which is almost always in alphabetical order.

A variety of models exist for the Yolov7 with different average accuracies and training time. The more accurate a model is, the higher the time and computational power required. Tests will be carried out using every model to investigate their compatibility with this project and if that extra accuracy that some models offer is worth the increased time commitment. The training weights for all of the models have been downloaded from the official repository and the models config files, which are already included in the Yolov7 package, need to be modified as to include the correct number of classes. The models that have been trained on images of size 640x640 pixels (plus the Yolov7-Tiny model) are referred to as P5 models because they support Yolov5 architectures and the ones that have been trained on images of size 1280x1280 are called P6 models and include an extra output layer for detecting larger objects.

Model	Test Size	AP^{test}	$AP_{50}{}^{test}$	$AP_{75}{}^{test}$	batch 1 fps	batch 32 average time
YOLOv7	640	**51.4%**	**69.7%**	**55.9%**	161 *fps*	2.8 *ms*
YOLOv7-X	640	**53.1%**	**71.2%**	**57.8%**	114 *fps*	4.3 *ms*
YOLOv7-W6	1280	**54.9%**	**72.6%**	**60.1%**	84 *fps*	7.6 *ms*
YOLOv7-E6	1280	**56.0%**	**73.5%**	**61.2%**	56 *fps*	12.3 *ms*
YOLOv7-D6	1280	**56.6%**	**74.0%**	**61.8%**	44 *fps*	15.0 *ms*
YOLOv7-E6E	1280	**56.8%**	**74.4%**	**62.1%**	36 *fps*	18.7 *ms*

Figure 5.6: The available Yolov7 models excluding Yolov7-tiny (source: Yolov7 repository).

Additions were made to the *detect.py* of the Yolov7 algorithm so that the number of frames each actor is present in is recorded. This information is then used to plot a bar plot comparing the screen time of each actor, from which conclusions like the hierarchy of the actors can be drawn.

Chapter 6

Results and Discussion

6.1 Overview

The main objective of this project was to design and implemented an algorithm which can train an object detection model that can successfully distinguish and track the actors featured in a film or TV series involving as little manual labor as possible. To achieve this a combination of several computer vision algorithms and techniques were used. In this chapter the results obtained from the designed implementation will be displayed and analysed with conclusions been drawn from them. A discussion will be held to debate whether the goals of the project have been achieved and possible future work will be considered.

6.2 Suitability of the face detection algorithms

The two algorithms employed to detect faces in images were Haar Cascade Classifiers and the Multi-Task Cascaded Convolutional Neural Network (MTCNN) face detection algorithm. Both algorithms were tasked with analysing 160 images collected from the web and short clips from TV episodes (80 images per actor) with only one face being present in each image. Out of these two algorithms, the Haar Cascades algorithm was notably faster when compared to the MTCNN algorithm however its performance was significantly lower. The former algorithm was not able to detect any faces in the majority of the images while the latter successfully identified at least one face in each image. Both algorithms incorrectly detected more than one face in several images, with the MTCNN algorithm having a lower ratio of multiple faces detected in the same image to images where at least one face was detected. The exact results of the tests conducted can be found in the table below.

	Haar Cascades	**MTCNN**
Time taken	∼53 seconds	∼714 seconds
Images where at least one face was detected	33	160
Images where multiple faces were detected	3	8

Table 6.1: Comparison of Haar Casscades and MTCNN detectors over 160 images

Overall it is clear that the MTCNN algorithm is better suited for this initial task because even though the Haar Cascades Classifiers algorithm is more than ten times faster than the MTCNN algorithm, it does not produce nearly enough results which are needed to train the face identification algorithm. Three different face identification algorithms were subsequently trained based on the data collected using the MTCNN. The algorithms are: Eigenfaces, Fisherfaces and Local Binary Pattern Histograms (LBPH). The time needed to train the face identification algorithm was extremely small and can be ignored for the most part. A noticeably large amount of data would be needed for the training time to be of significance.

6.3 Choosing the algorithm to automatically label the images

A total of 4520 images were collected from a few episodes of a TV series using the script implemented in the previous chapter. The face identification algorithms trained previously were then tasked with automatically labelling each face detected. Face detection was done with both Haar Cascade Classifiers and the MTCNN algorithm. To minimise false-positive faces detected and ensure that the faces kept contained the correct actor, the threshold value of 60 was implemented meaning that if the distance supplied by the face identification algorithm for a particular face was above the threshold, then the face was discarded. For reference, if we were to assume that an average person can label an image in about 8 seconds, it would take them around 10 hours to go through the whole dataset. The results of the test can be found in the table below.

	Eigenfaces	**Fisheraces**	**LBPH**
Time taken with Haar Cascades (in seconds)	∼2033	∼2187	∼1749
Time taken with MTCNN (in seconds)	∼13278	∼15334	∼17571
Labelled images kept with Haar Cascades	0	411	226
Labelled images kept with MTCNN	98	3139	1658

Table 6.2: Comparison between the three face identification algorithms.

Although the time required to process the dataset by all three algorithms is notably faster then the assumption we made above, the processing time does not appear to be a deciding factor when comparing these algorithms. Based on the tests conducted, it is apparent that the Eigevalues algorithm is not suitable for this project. It wasn't noticeably faster than the other algorithms and it performed extremely poorly in every test conducted. The Fisherfaces algorithm surprisingly kept more images than the LBPH algorithm which was very unexpected based on the research done during the implementation part of the project. Upon closer inspection of the images kept by both algorithms, the Fisherfaces one saved far more false-positives than the LBPH with some of them missing a human entirely. The dataset kept by the LBPH algorithm was also flawed but not to the same degree. After careful consideration, it was decided to prioritize quality over quantity thus the algorithm chosen as the most suitable for this part was Local Binary Pattern Histograms. An augmented copy of

each image was then created (apart from some augmentations that failed) bringing the size of the training database up to 3274.

6.4 Using the training data to detecting actors in videos

The database containing the 3274 images was split into the three required folders with a 70%-20%-10% ratio, as per the recommendation of the authors of the algorithm which were subsequently moved to the data folder located in the Yolov7 directory. The new data .yml file was also created specifying the folder paths, number of classes and class names.

```
train: ./data/train
val: ./data/val
test: ./data/test

# number of classes
nc: 2

# class names
names: ['Jesse Pinkman', 'Walter White']
```

Listing 6.1: Custom data .yaml file made to detect the two actors.

6.4.1 Training the standard Yolov7 model using the automatically label dataset

The standard model of the Yolov7 algorithm was then trained using the subsequent command.

```
python train.py --workers 16 --device 0 --batch-size 32 --epochs 300
--img 640 640 --data data/custom_data.yaml --hyp data/hyp.scratch.custom.yaml
--cfg cfg/training/yolov7-custom.yaml --name yolov7
--weights weights/yolov7_training.pt
```

Listing 6.2: Training command used for standard Yolov7 model.

Each parameter supplied is explained below.

- **Workers:** Parallel computing units used to speed up the processing of the algorithm. Each worker is responsible for processing a portion of the input data and the output from each one is combined to produce the final result.
- **Device:** Specifies the hardware to be used for training. The GPU is used when it is set to a number (multiple numbers means multiple GPUs) and the CPU is used when the parameter given is 'cpu'. Object recognision algorithms are almost always trained on a GPU rather than a CPU due to the incredibly higher speeds.

- **Batch-size:** It determines the number of samples the model will process in one forward/backward pass. If the batch size is set to 32, the model will process 32 training samples at a time before updating its internal parameters based on the error calculated from the predictions.
- **Epochs:** An epoch refers to one complete pass through the entire training dataset.
- **Img:** Resize the image to the dimensions specified.
- **Data:** The file specifying the training database paths, number of classes and their names.
- **Hyp:** Hyperparameters are used to control various aspects of the model's training process such as the learning rate, momentum, weight decay and box loss gain. The hyperparameter file supplied is used without been modified.
- **Cfg:** A configuration file that contains settings and parameters used to configure the particular model. Each Yolov7 has their own configuration file which is included with the rest of the algorithm. The number of classes have been altered in the file to match the data file created.
- **Name:** The name of the folder the results of the training process will be stored in.
- **Weights:** Learned parameters of a model that are updated during the training process. The initial training weights for each model can be downloaded from the official repository.

6.4.2 Observations

The massive computation requirements required to train a Deep Neural Network became apparent when the command was executed for the first time on a local machine. The machine which was equipped with a ninth generation Intel-i5, 16GB of RAM and a Nvidia GeForce RTX 2060 was not sufficient for training this model. More specifically, the GPUs' RAM, 8GB of dedicated RAM, was not enough (out of memory error). The local machine was only able to run the training script when the number of workers and the batch size was reduced to 4 which would result in extremely low speeds and accuracy score as such a low batch size makes the algorithm converge at a much slower rate needing many more epochs to reach a high accuracy score. Training the algorithm with the CPU would provide double the amount of RAM, however it would result in even longer training times as a CPU is not designed for parallel computing of this scale. Options such as Google Colab were considered however to train the Yolov7 basic model with the recommended hyperparameters, a University of Sheffield's High Performance Computer (HPC) was eventually used. The HPC used for this test was Bessemer which is equipped with Intel Xeon Gold 6138 (2.00GHz) CPUs and NVIDIA Tesla V100 GPUs, each one with 32GB of GDDR5 memory.

The model was trained on Bessemer once using a CPU and once using a GPU. The different hardware used had a very small impact on the resulting accuracy of the algorithm as the GPU version had an overall Mean Average Precision of 0.69 on an IoU of 0.5 (mAP@.5) and 0.502 across multiple IoU thresholds ranging from 0.5 to 0.95 with an increment of 0.05 (mAP@.5:.95) while the one trained on the CPU had a mAP@.5 of 0.705 and a mAP@.5:.95 of 0.501. The above results were acquired from the in-build test function provided by the Yolov7 algorithm are not guaranteed to accurately represent the model. For reference, the basic Yolov7 model has mAP@.5:.95 of 51.2 on the COCO dataset trained on an Nvidia Tesla V100 GPU according to the same functionality. The biggest difference was the training time required as the GPU needed 7.5 hours for 300 epochs while the CPU needed 111 hours for the same amount of epochs. Although the training times are not guaranteed, especially on a publicly accessible High Performance Computer, the GPU was more than ten times faster than the CPU. The results obtained from this test have proven that our automatic labelling algorithm is able to produce results that are on par with other Deep Neural Networks when it comes down to computational requirements without sacrificing accuracy. A high power GPU such as the aforementioned Tesla V100 would be needed to train any Deep Neural Netowrk in a reasonable amount of time however renting GPU for the purpose of training neural networks is easier than ever. Several cloud computing services such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure provide GPU instances that can be rented by the hour or minute for relatively low cost. Manual tests of varying difficulty were subsequently undertaken.

The first test conducted involved the detection of the two desired actors in a relatively simple setting. The clip used took part in a single room with only the desired actors appearing and consisted of mostly close up shots in a brightly illuminated and unobscured environment. The clip with a length of 3 minutes and 50 seconds was processed using an Nvidia GeForce RTX 2060 (the same one mentioned above) with the best weights obtained by the standard Yolov7 model and an image resolution of 640x640 in 806 seconds or roughly 13 minutes. Based on the results obtained from the other tests conducted on a HPC, it is safe to assume that the overall processing time would be reduced dramatically on a higher-end GPU. The model was able to successfully identify and distinguish the two actors in most of the close-up shots but would occasionally misidentify an actor or identify them multiple times as different actors. This phenomenon mostly occurred when the actor's face was party obscured, when their face was shaded or only their face profile was visible and may be solved by expanding the training dataset and ensuring a wide range of angles are captured for each actor. The algorithm would also occasionally struggle with some of the shots from further away but that maybe due to the threshold value set in place. A threshold value of 0.50 was set to limit the number of false identifications but as a result some correct detections could also have been rejected. The best solution would once again be to introduce more variety to the training set. A final observation was that some facial accessories (i.e. glasses and mask) were confidently identified as the person that wears them most often even when they were not located on their face. The two possible explanations are that those objects appear in the majority of

the images of that actor or that training on too many epochs have cause overfitting (occurs when a statistical model fits exactly against its training data).

Figure 6.1: Frame from the first test.

The second test aimed to evaluate the trained model's performance when tasked with identifying those 2 specific actors in more complex environments. The majority of the scenes were either very crowded or only featured irrelevant actors, some of which closely resembled the desired ones. The whole clip was just short of 5 minutes and was processed in 1198 seconds or roughly 20 minutes using the same parameters as before. The test revealed a major flaw with the algorithm. Although the model was able to almost always successfully identify the desired actors in a crowded setting with high accuracy, it would react rather unpredictably when detecting another person. It seemed that it would randomly assign that person a class with semi-high probability effectively nullifying the data returned at the end of the video analysis. The most plausible explanation is that the training data used to train the model did not have clear labels for the desired classes. Another possible cause of this could be that the hyperparameters have not been tuned or optimized for this specific problem. The best possible cause of action may be to once again increase the size of the training data in hopes of reducing the confidence score assigned to the irrelevant actors below the threshold. The other possible approach would be to fine-tune the model but the whole process requires a vast amount of experimentation and time.

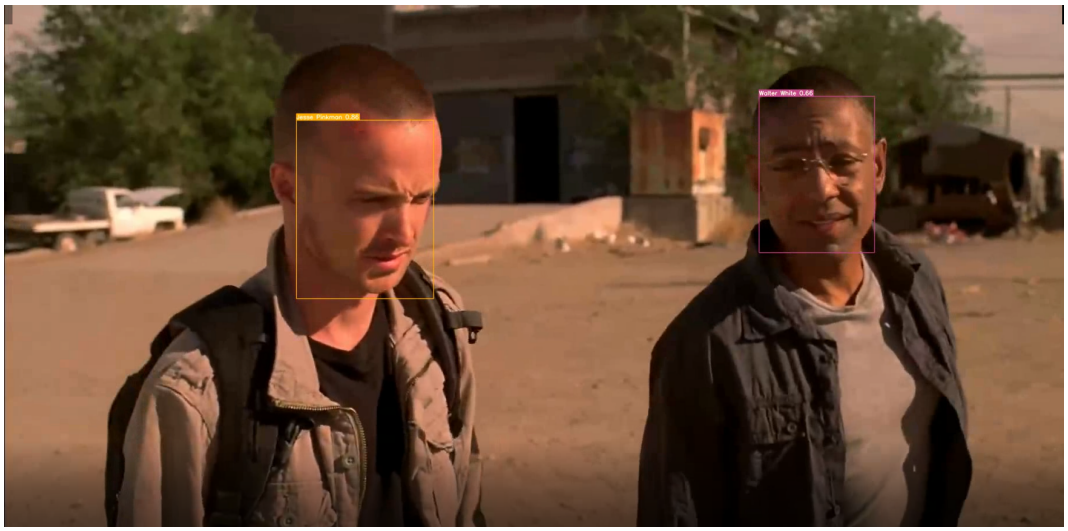

Figure 6.2: Frame from the second test.

The third and final test was intended to test the performance of the algorithm when the actors' facial characteristic have been altered significantly and no such samples were present in the training data. In this case, an clip was chosen in which the desired actors had aged, had altered both hair and facial hair and had developed several scars and wounds on their faces. The model was able to correctly identify the actors with high accuracy from close-up shots but started to struggle after that. These results were not completely unexpected thus the algorithm was designed in a way that new data can be easily added to the training set so that the model can be re-trained and improved as time goes on and a series advances.

Figure 6.3: Frame from the third test.

6.4.3 Evaluating the different models on the same dataset

To determine the suitability of the various Yolov7 models for this project, they were all trained on the first dataset consisting of only two actors and their performance was compared based on accuracy, computational requirements, and training time. The recommended parameter values by the Yolov7 algorithm authors and the original hyperparameters were used for the training. Each model was trained on a GPU with 8 workers for 300 epochs, using their corresponding initial training weights. However, the training was conducted on the University of Sheffield's new high performance computer, Stanage, which is equipped with an Nvidia A100 with 80GB of RAM. This is a big improvement from the Nvidia V100 with 32GB of RAM used by Bessemer, as the A100 is 60%-70% faster according to the manufacturer's specifications. By conducting the training on a faster GPU, the training time and computational requirements of each model were reduced resulting in more accurate performance measurements.

The P5 models (Yolov7, Yolov7-X and Yolov7-Tiny) were trained on images with size 640 by 640 and a batch size of 32 while the P6 models were trained on images with size 1280 by 1280 and a batch size of 8. Once again, the average time calculated for each model should be treated as a rough estimate and not a guarantee. Regarding the testing functionality, all models were tested on the Stanage HPC using their respective best weights, a batch size of 32 an IoU of 0.65 and a confidence threshold of 0.001.

Model	**Image size**	**mAP@.5:.95**	**Average time**	**Model training time**
YOLOv7	640	0.537	2.0 ms	2.520 hours
YOLOv7-Tiny	640	0.553	1.6 ms	8.990 hours
YOLOv7-X	640	0.544	2.6 ms	8.305 hours
YOLOv7-W6	1280	0.574	4.5 ms	6.755 hours
YOLOv7-E6	1280	0.582	9.7 ms	9.306 hours
YOLOv7-D6	1280	0.589	10.9 ms	10.628 hours
YOLOv7-E6E	1280	0.602	27.0 ms	12.782 hours

Table 6.3: Comparison of the different YOLO models.

The Mean Average Accuracy of the different models are on par with the ones supplied by the authors of the algorithm but there are some discrepancies between the average frame processing time acquired from the ones given, most likely a result of the different hardware used to train the models. The P5 models offer very similar accuracy and hardware requirements and mainly differ in model training time and frame processing time. The Yolov7-Tiny seems to be the best option from the P5 models as it offers the highest accuracy and lowest frame processing time which can be more important than the model training time since the model will only be trained a few times while it will process many more videos. The P6 models require more than double the amount of memory used by the P5 models due to

the higher resolution of the images even with a smaller batch size being requested. Out of the P6 models, although the Yolov7-E6E offers the highest accuracy, its frame processing time is significantly longer than the rest and does not justify the increase in accuracy. The rest of the P6 models are all viable options and are interchangeable at the discretion of the user and what they feel is more suitable for their model.

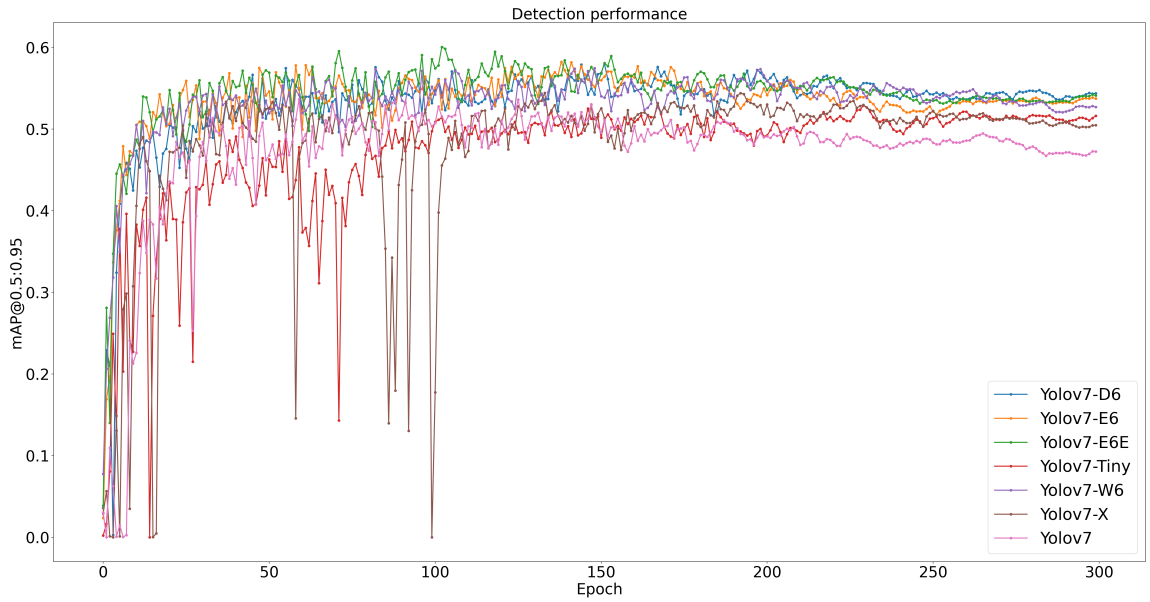

Figure 6.4: The Mean Average Precision of every model.

6.4.4 Performance evaluation with increased number of classes and decreased number of training images

To test the limits of the algorithm, we wanted the trained model to detect 6 actors using 10 images per actor to train the face classification algorithm and clips from a single episode as the training set for the Deep Neural Network (resulting in less images per actor). The training and detection times were not affected significantly by the increased classes and were withing the acceptable limits we would expect from an algorithm of this scale. The model was surprisingly able to correctly identify the requested actors with a higher than expected confidence score from small and medium distances but obviously struggled with most other shots. The problem with the misclassified irrelevant actors was present in this model as well, which was not unexpected, and posses a real threat to the project. More tests would need to be conducted to uncover the source of this error and the best approach to fixing it.

Figure 6.5: Frame from the fourth test.

6.4.5 Gathering data and drawing conclusions

When a video has been successfully processed by the chosen model, information such as the number of frames each actor appears in or in how many frames a pair of actors are on screen together can be collected, processed and visualised. With an accurate enough model, one could calculate the precise screen time for any actor and draw conclusions about their role in the movie (protagonist, deuteragonist, supporting character, etc.). It is clear that the results obtained from the models we trained are not reliable enough in most cases to produce accurate results but this functionality will most likely prove useful after further experimentation.

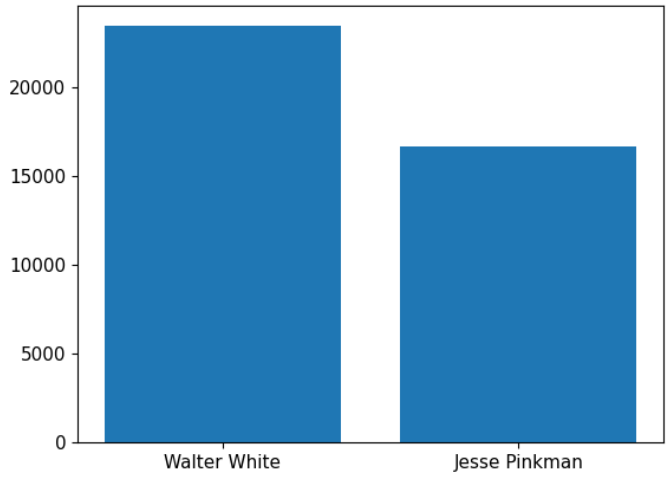

Figure 6.6: Bar chart indicating the number of frames each actor appears in.

6.4.6 Further work

The biggest flaw in the system developed in this paper is the misidentification of irrelevant actors in the media processed leading to inaccurate results and conclusions. Further experimentation is needed to establish if the source of this flaw is indeed the lack of sufficient training data as speculated and how much would it cost in regards to computational time and requirements to fix it.

A possible addition to the algorithm created that is worth exploring is the automation of fine-tuning a model's hyperparameters. The algorithm can be programmed to repeatedly train itself while slowly adjusting its hyperparameters and storing those resulting in a higher performance score by the build-in test functionality.

Chapter 7

Conclusions

In this paper we have managed to develop an algorithm that can successfully train an object detection model to recognise and track specific individuals in an image or throughout a video stream that includes multiple continuous shots, while also accounting for changes like different camera orientations, lighting, and weather conditions and simultaneously minimising the amount of human intervention needed.

A human operator supplies individual images of each actor and time frames in videos that feature said actors. Just from those inputs, the algorithm implemented is able to create and label a large training database before training the object detection model, achieving an accuracy of up to 0.60 mAP@.5:.95. Manual tests indicate that the model trained using the algorithm performs quite well, especially in close-up shots, and can process all types of media in a reasonable amount of time for a project of this scale. After processing the supplied media it is also returns data that it had gathered regarding the actors like in how many frames each one was present.

The project is considered successful as most of the objectives set out in the beginning have been met and a plan has been formulated to address the rest. The algorithm has automated some tasks which were previously done by humans and were extremely time-consuming, labour-intensive and mundane without causing a significant drop in the performance of the subsequently trained model. Tests conducted on videos where the characters had significantly aged proved that the trained model is able to use a combination of facial features to identify the individuals as the confidence average had dropped but not by a massive amount. The algorithm was also able to collect and label the training dataset extremely fast compared to the amount of time it would have taken a human to do it and the time required to train the model was within the limits of what you would expect when training a Deep Neural Network. The final objective was not met fully but ideas exists as to how to fulfil it. The current problem is that the trained model falsely assigns random labels to the actors we are not interested in thus making the data collected unusable.

A larger dataset needs to be collected and a new model needs to be trained in order to improve

the algorithms precision and establish if the lack of sufficient training data is the cause of the incorrect detections. Another idea worth exploring once the current problems have been solved is the automation of fine-tuning the hyperparameters of the model, a process known for requiring massive amounts of time and experimentation. The algorithm could be tasked with continuously re-training the model and slowly tweaking the hyperparameters, saving the ones that wield the best results.

Bibliography

Ahsan, M. M., Li, Y., Zhang, J., Ahad, M. T., and Gupta, K. D. (2021). Evaluating the performance of eigenface, fisherface, and local binary pattern histogram-based facial recognition methods under various weather conditions. *Technologies*, 9(2).

Bartholomew, D. (2010). Principal components analysis. *International Encyclopedia of Education (Third Edition)*, pages 374–377.

Belhumeur, P., Hespanha, J., and Kriegman, D. (1997). Eigenfaces vs. fisherfaces: recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):711–720.

Bochkovskiy, A., Wang, C., and Liao, H. M. (2020). Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934.

Bridle, J. S. (1989). Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. *Proceedings of the 2nd International Conference on Neural Information Processing Systems*, page 211–217.

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1993). Signature verification using a "siamese" time delay neural network. *Proceedings of the 6th International Conference on Neural Information Processing Systems*, page 737–744.

Budiman, A., Fabian, Yaputera, R. A., Achmad, S., and Kurniawan, A. (2023). Student attendance with face recognition (lbph or cnn): Systematic literature review. *Procedia Computer Science*, 216:31–38. 7th International Conference on Computer Science and Computational Intelligence 2022.

Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M., and Kalinin, A. A. (2020). Albumentations: Fast and flexible image augmentations. *Information*, 11(2).

Cubuk, E. D., Zoph, B., Mané, D., Vasudevan, V., and Le, Q. V. (2018). Autoaugment: Learning augmentation policies from data. *CoRR*, abs/1805.09501.

Dai, J., Li, Y., He, K., and Sun, J. (2016). R-FCN: object detection via region-based fully convolutional networks. *CoRR*, abs/1605.06409.

Deng, J., Guo, J., Zhou, Y., Yu, J., Kotsia, I., and Zafeiriou, S. (2019). Retinaface: Single-stage dense face localisation in the wild. *CoRR*, abs/1905.00641.

Evgeniou, T. and Pontil, M. (2001). Support vector machines: Theory and applications. *Support Vector Machines: Theory and Applications*, 2049:249–257.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Fu, C., Liu, W., Ranga, A., Tyagi, A., and Berg, A. C. (2017). DSSD : Deconvolutional single shot detector. *CoRR*, abs/1701.06659.

Fukushima, K. and Miyake, S. (1982). Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, 15(6):455–469.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587.

Girshick, R. B. (2015). Fast R-CNN. *CoRR*, abs/1504.08083.

He, K., Zhang, X., Ren, S., and Sun, J. (2014). Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

Huang, Q., Xiong, Y., and Lin, D. (2018). Unifying identification and context learning for person recognition. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Jayaswal, R. and Dixit, M. (2020). Comparative analysis of human face recognition by traditional methods and deep learning in real-time environment. *2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT)*, pages 66–71.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. *Advances in Neural Information Processing Systems*, 2.

Lin, D., Kapoor, A., Hua, G., and Baker, S. (2010). Joint people, event, and location recognition in personal photo collections using cross-domain context. *Computer Vision – ECCV 2010*, pages 243–256.

Lin, T., Dollár, P., Girshick, R. B., He, K., Hariharan, B., and Belongie, S. J. (2016). Feature pyramid networks for object detection. *CoRR*, abs/1612.03144.

Lin, T., Goyal, P., Girshick, R. B., He, K., and Dollár, P. (2017). Focal loss for dense object detection. *CoRR*, abs/1708.02002.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C., and Berg, A. C. (2015). SSD: single shot multibox detector. *CoRR*, abs/1512.02325.

Long, J., Shelhamer, E., and Darrell, T. (2014). Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038.

Neubeck, A. and Van Gool, L. (2006). Efficient non-maximum suppression. *18th International Conference on Pattern Recognition (ICPR'06)*, 3:850–855.

Norvik, G. (2022). Facial recognition techniques comparison for in-field applications : Database setup and environmental influence of the access control. *n/a*, page 32.

Ojala, T., Pietikainen, M., and Harwood, D. (1994). Performance evaluation of texture measures with classification based on kullback discrimination of distributions. *Proceedings of 12th International Conference on Pattern Recognition*, 1:582–585 vol.1.

Ojala, T., Pietikäinen, M., and Harwood, D. (1996). A comparative study of texture measures with classification based on featured distributions. *Pattern Recognition*, 29(1):51–59.

Paz Castro, I. (2016). Mtcnn: Multi-task cascaded convolutional networks.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788.

Redmon, J. and Farhadi, A. (2016). YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242.

Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *CoRR*, abs/1804.02767.

Reitz, K. (2011). Requests: Http for humans™. `https://requests.readthedocs.io/en/latest/`.

Ren, S., He, K., Girshick, R. B., and Sun, J. (2015). Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497.

Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *n/a*.

Sirovich, L. and Kirby, M. (1987). Low-dimensional procedure for the characterization of human faces. *J. Opt. Soc. Am. A*, 4(3):519–524.

Sumengen, B., Lee, K., Gokturk, S., and Anguelov, D. (2007). Contextual identity recognition in personal photo albums. *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7.

Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition.*

Tharwat, A., Gaber, T., Ibrahim, A., and Hassanien, A. E. (2017). Linear discriminant analysis: A detailed tutorial. *Ai Communications*, 30:169–190,.

Toudjeu, I. and Tapamo, J.-R. (2019). Circular derivative local binary pattern feature description for facial expression recognition. *Advances in Electrical and Computer Engineering*, 19:51–56.

Turk, M. and Pentland, A. (1991). Face recognition using eigenfaces. *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 586–591.

Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 1:I–I.

Wang, C.-Y., Bochkovskiy, A., and Liao, H.-Y. M. (2022). Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *n/a.*

Yu, W., Yang, K., Bai, Y., Yao, H., and Rui, Y. (2014). Visualizing and comparing convolutional neural networks. *CoRR*, abs/1412.6631.

Zhang, K., Zhang, Z., Li, Z., and Qiao, Y. (2016). Joint face detection and alignment using multi-task cascaded convolutional networks. *CoRR*, abs/1604.02878.