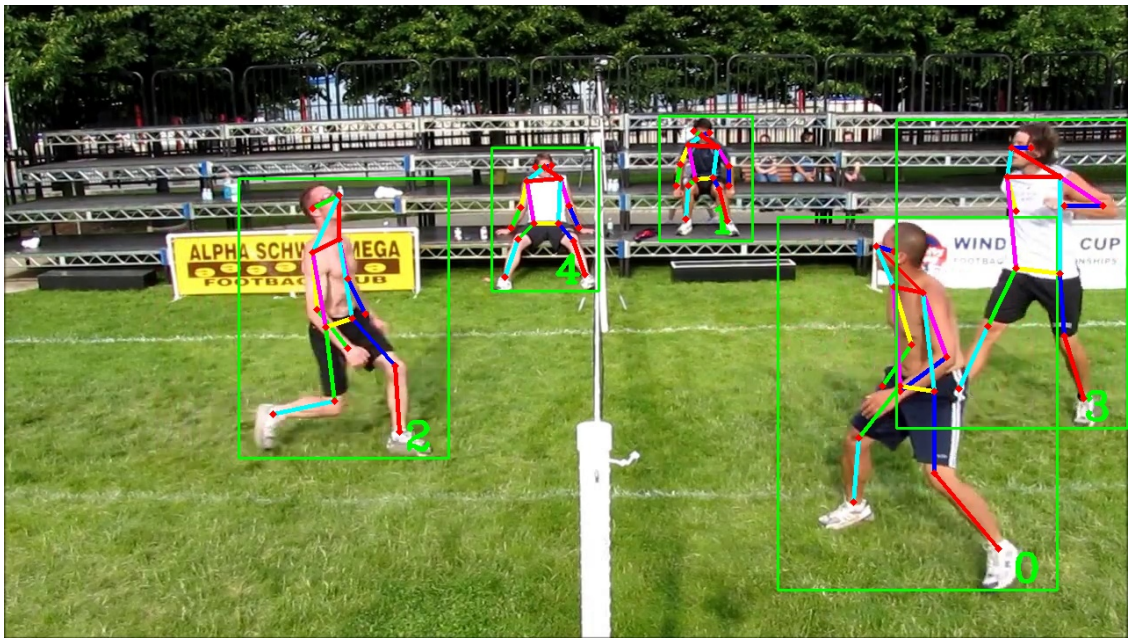
CHALMERS
UNIVERSITY OF TECHNOLOGY



Optimized pose tracking on edge devices

Master's thesis in System, Control and Mechatronics

THANISORN SRIUDOMPORN

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Optimized pose tracking on edge devices

THANISORN SRIUDOMPORN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2022

Optimized pose tracking on edge devices
THANISORN SRIUDOMPORN

Supervisor: André Dankert, Knightec AB
Examiner: Yasemin Bekiroglu, Department of Electrical Engineering

Master's Thesis 2022
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Human pose estimation and Human pose tracking predicted by Neural networks and Tracking algorithms perform on Nvidia Jetson Nano as an edge device.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Optimized pose tracking on edge devices
THANISORN SRIUDOMPORN
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The purpose of this master thesis is to evaluate the performance of different models for human pose tracking to serve edge computing purposes. In comparison to cloud computing, edge computing performs the computation on edge devices, which has less computing power than servers. Two tasks are tested on the edge device by four different models: Flow Track, Light Track, Joint Flow, and Face-to-pose track. The ability to estimate human pose on sequence images of each model is evaluated by the Multiframe person pose estimation task. The Multi-person pose tracking task shows how accurate the models are for tracking human pose. Posetrack dataset and Coco dataset are used to train models. The mAP (Mean average precisions) and the MOTA (Multiple object tracking accuracies) are calculated to measure the performance of each model on the custom dataset in multi-frame person pose estimation task and multi-person pose tracking task, respectively. The FPS (Frames per second) is measured to measure the speed of each model. The mAp changes from 41.2% for JointFlow to 81.3% for LightTrack, and the FPS changes from 0.29 for JointFlow to 1.77 for FlowTrack in the multi-frame person pose estimation task. The MOTA changes from 21.5% for JointFlow to 42.5% for FlowTrack. In the multi-person pose tracking task, the fastest model and the slowest model are FlowTrack at 0.73 FPS and JointFlow at 0.29 FPS. The results show that the top-down models outperforms the bottom-up models.

Keywords: Pose Tracking, Pose Estimation, Multiple-Object Tracking, Edge computing, Edge device, Image Analysis, Neural Network, Computer Vision, Deep Learning, Modeling.

Acknowledgements

This thesis would not have been possible without the support of many people. I would like to thank my supervisors at Knightec AB, André Dankert, and Alex Darborg, for their suggestions and supports in this thesis.
In addition, I would like to express my gratitude to great people in Thailand, Boonchai, Bantawan, Thunradee, and Pattraporn. They have been a great source of support.
I would also like to thank my Thai friends in Gothenburg, Apichai, Papop, Wasinee, and Nuttida for their help in collecting data for the dataset of this thesis.
Finally, I would like to express my appreciation to my examiner, Yasemin Bekiroglu, for her guidance and feedback.

Thanisorn Sriudomporn, Gothenburg, August 2021

Contents

List of Figures

List of Tables

1
Introduction

In the last few years, human pose tracking has gained more attention due to its broad applications. For example, human pose tracking can be used in risk prediction for humans and machines for safety purposes. In addition, the unmanned store is also a new trend. Cameras track each customer and identify purchases from the customer's pose and position.
Similar to human pose estimation, human pose tracking combines object detection, human pose estimation, and multiple object tracking. The model should detect every human in each frame, estimate poses, and assign each pose an identity number. The example of human pose tracking is shown in Figure 1.1. However, the available solutions are prone to errors and require expensive computation. In addition, some extreme positions where human joints are not visible in the frame could lead to fault detection in tracking. Another issue worth mentioning is that most models are trained and tested on a high-performance computer. However, the performance of these models on less capable devices is still questionable.

Figure 1.1: An example of human pose tracking in an image.

The edge computing research field also has been on a rising trend. The main idea of edge computing is to employ edge devices for computation rather than servers or clouds. Small and less power-consuming computers are often the popular choice for edge devices, which are available on-site.
A combination of pose tracking and edge computing can be applied to many applications, such as tracking human activity, giving advice, or warning based on human

gestures. As an example, exercise applications track human poses and give advice or warning based on human gestures. However, edge devices are low-performance devices compared to desktop computers or servers. Therefore, it is an open research area to perform pose tracking or similar applications with good accuracy and speed using edge devices.

1.1 Aim

This thesis aims to propose a pose tracking approach and evaluate on edge devices with existing approaches. The research questions which this thesis will address are:

- How well can the state-of-the-art top-down pose tracking approaches estimate and track human pose based on measurements of average precision (AP) and multiple object tracking (MOT) on edge devices compared to state-of-the-art bottom-up pose tracking approaches?
- How fast, in terms of frames-per-second (FPS), can state-of-the-art top-down pose tracking approaches be used to track human pose on edge devices compared to state-of-the-art bottom-up pose tracking approaches?
- What is the most efficient approach to optimize and improve both accuracy and speed of pose tracking on edge devices?
- What is the effect of speed, in terms of frames-per-second, to the accuracy, based on the visualization from edge device and USB camera in real-time, where image acquisition and pose tracking are performed simultaneously?

The predictions of each approach is benchmarked in the same as the PoseTrack challenge on a custom dataset. Finally, the comparison between each approach and the performance of less capable devices is discussed.

1.2 Limitations

There are several formats of human pose annotation for human pose tracking. In this thesis, the human pose is represented by 15 key points: nose, left ear, right ear, left shoulder, right shoulder, left elbow, right elbow, left wrist, right wrist, left hip, right hip, left knee, right knee, left ankle, and right ankle.
Nividia Jetson Nano is selected to represent the edge device in this thesis. Performances, which are reported in this thesis, are measured on Nvidia Jetson Nano.
Offline pose tracking approaches, which process a batch of frames at once, are not evaluated in this thesis since the thesis aims to propose a novel approach in real-time.

2

Background

2.1 Convolutional Neural Networks - CNNs

In the last few years, convolutional neural networks (CNNs) play a significant role in object detection, human pose estimation, and similarity prediction. The definition, components, and training procedure of CNNs are briefly introduced in this section. In conventional machine learning techniques, features need to be extracted and selected as a feature engineering step before the models can be trained. However, selecting the proper features for each model's goal is complicated. The feature selecting problems are solved by implementing deep learning techniques. The deep learning models extract essential features from the input without manual feature selection needed. Deep learning models are called differently according to their architectures. For example, Recurrent neural networks [1] contain blocks of artificial neural networks that can be repeated. Another example is Long Shot Term Memory Networks [2] which are the neural networks that save previous predictions in memory. Figure 2.1 represents the architectures of Recurrent neural networks and Long Shot Term Memory Networks.

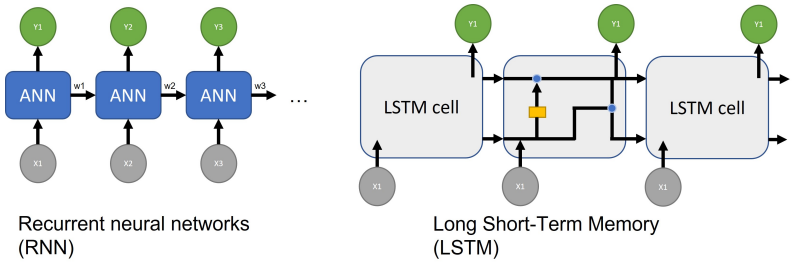


Figure 2.1: Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) are the Deep learning model examples. Both of them have different architectures and different names for different purposes.

The CNNs is the hierarchical feature detector with feed-forward architecture as described in [3]. Natural language processing [4], signal processing [5], object detection [6], and medical image processing [7] are popular applications of CNNs. Figure 2.2 shows an example of the architecture of CNNs. Each particular neuron in CNNs is the artificial neural. It takes vector X as input and gives Y as an output of function F with weight vector W. This relation is represented as in equation (2.1) [8]. The trained weight vector is used to perform feature extraction.

$$F(X, W) = Y \tag{2.1}$$

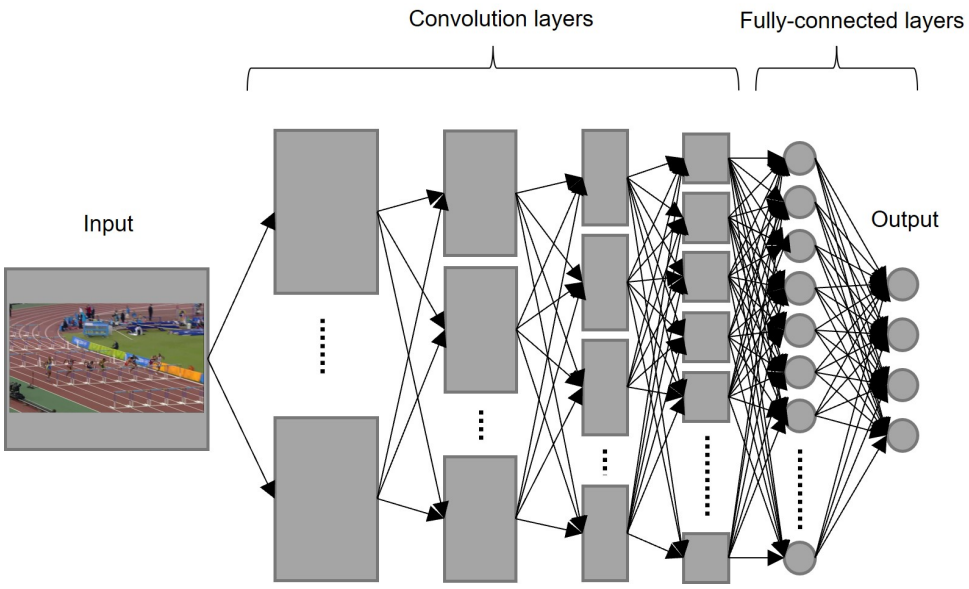


Figure 2.2: The basic architecture of CNNs consists of the convolutional layers and the fully-connected layers. In addition, in some models, the pooling layers are included in the architecture.

The general models of CNNs have four different types of hidden layers. Each layer is stacked to each other to make the model deeper. The abstract features are extracted in the deeper layer compared to the regular feature [9]. The four types of hidden layers for CNNs are described in subsection 2.1.1.

2.1.1 Layers

2.1.1.1 Convolutional layer

The convolutional layer is the reason why this type of architecture is called a Convolutional Neural Network. Biological behavior of animal inspires the design of this layer [10]. Some animal, such as cat, have area that can decode colors. Each region in the input, called a receptive field, performs a convolution operation with the weight vectors. The result of the convolution operation C_{out} is introduced to the non-linear function [11]. Equation (2.2) shows the formula which is applied to each region. The weight vectors are also known as kernels or filters.

$$C_{out} = \sigma((W * X) + b) \tag{2.2}$$

where X is the input region, W is the filter, b is the bias, $*$ is the convolution operation, and σ is non-linear function applied in this layer.
The convolution operation is performed slidingly, both vertical and horizontal, all over the input vector. Figure 2.3 illustrates how the convolutional layer works.

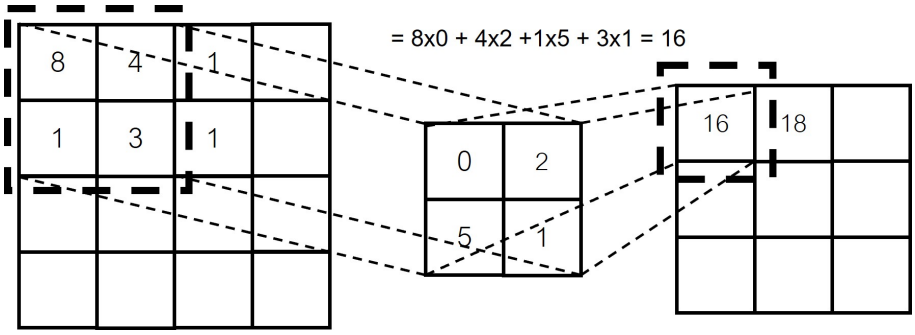


Figure 2.3: In each slide, the numbers in the kernel are multiplied by the number in the matrix slide. Then, the sum of the multiplication is placed into the output matrix. This operation is performed for every slide in the input matrix.

After the filter is applied over the input, the output vector from this layer is a H x W x D shape vector. H and W are the output dimensions, and D is the number of filters. The output dimension D_{out} is calculated from the kernel size K, stride S, zero padding P, and input dimension D_{in} as shown in equation (2.3).

$$D_{out} = \frac{D_{in} - K + 2P}{S} + 1 \tag{2.3}$$

The few first layers extract simple features such as vertical edges or horizontal edge. The complex features such as eyes or human are detected in the deep layer [10].

2.1.1.2 Pooling layer

Pooling is the operation for down-sampling to simplify data. The pooling operation can be seen as the resolution reduction between convolutional layers. The pooling layer works similarly to the convolutional layer. The pooling function is applied to the input as a filter. There are several pooling functions. The well-known functions are max-pooling and average-pooling. The max-pooling draws the maximum value in each kernel and forms the new matrix for further layers. Similarly, the average-pooling layer calculates the mean value instead of the maximum value. The example of max-pooling and average-pooling is demonstrated in Figure 2.4. On the top left kernel marked in green, six is given due to the max-pooling while the average-pooling provides three. The benefits of using the pooling layer are to lessen the number of parameters of the model and prevent overfitting.

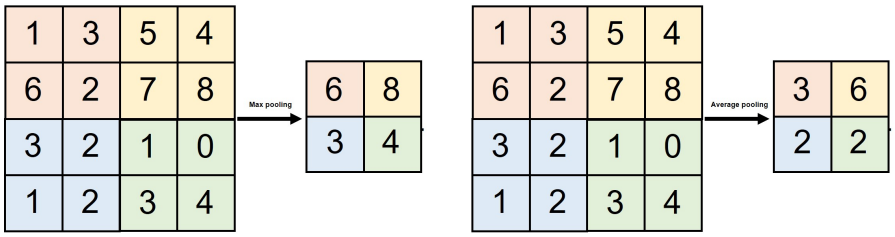


Figure 2.4: In each kernel, the max-pooling selects the highest number and place it in the output, while the average-pooling calculates the mean of all numbers and send it to the output.

2.1.1.3 Fully connected layer

The feed-forward neural networks are called fully connected layers in CNNs. Each node in the layer is connected to every node in the previous and subsequent layers. The calculation in the connection is simple. First, the input is multiplied by a weight. Then, a bias is added. Figure 2.5 shows the relationship of each fully connected layer. Typically, fully connected layers are placed at the end of the network. Thus, the output of the last pooling layer must be flattened before sending into the first fully connected layer. The disadvantage of the layer is that each layer increase number of trainable parameter. The dropout layer is introduced to get rid of some nodes and connections randomly.

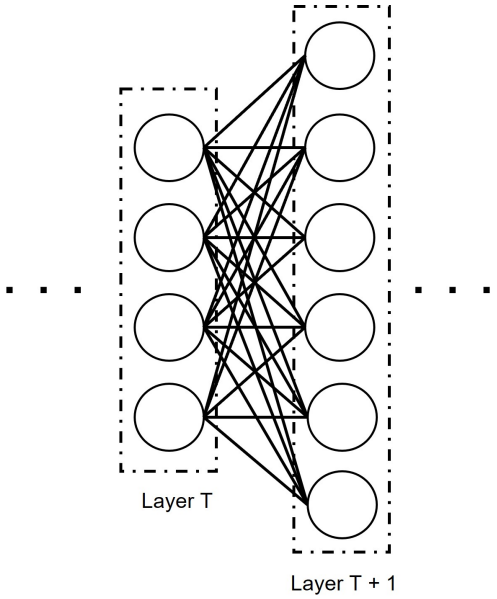


Figure 2.5: Every node in layer T connects to every node in later T + 1 with weights and bias, represented with the black lines in this figure.

2.1.1.4 Activation Function

The activation function is used with the output of each layer to introduce nonlinearity to the network. The commonly used functions are the Rectified Linear Unit (ReLU) [12] and Sigmoid [13]. The ReLU function is better and faster to process than the Sigmoid function in image processing applications due to its complexity [14]. Equation (2.4) and Equation (2.5) represent the ReLU function and Sigmoid function.

$$f(x) = max(0, x) \tag{2.4}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

The comparison between plots of ReLU function and Sigmoid function is shown in Figure 2.6.

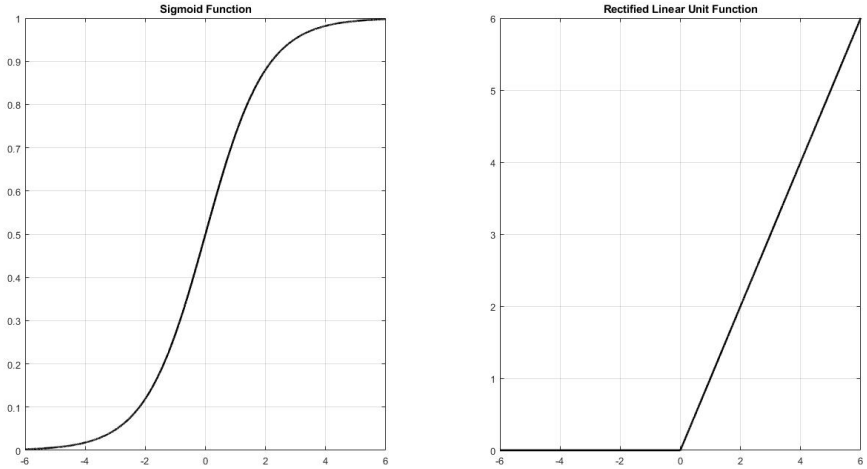


Figure 2.6: The plots show the values of the Sigmoid function and the ReLU function when the input is varied between -6 to 6.

2.1.2 Training

The training process means updating parameters to maximize the network's performance. In the first step, the image is passed through each layer to get a prediction. This step is called Forward propagation. Then, the loss, which represents the error between ground truth (Y_i) and prediction($\hat{Y}_i$), is calculated. Mean squared error (MSE), shown in Equation (2.6), is an example of loss.

$$Loss = MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.6}$$

Finally, the trainable parameter is updated from the derivation of the loss function in the Backward propagation step. There are several ways to update the parameter, such as Basic gradient descent [15], Adam optimizer [16], or RMSprop [17]. The gradient descent equation is shown in Equation (2.7) as an example where θ_i is the parameter in each layer, γ is the learning rate, and $J(\theta)$ is the loss function. The learning is the hyperparameter that is to be tuned. The overview of the training process is demonstrated in Figure 2.7.

$$\theta_i = \theta_i - \gamma\frac{\partial J(\theta)}{\partial \theta_i} \tag{2.7}$$

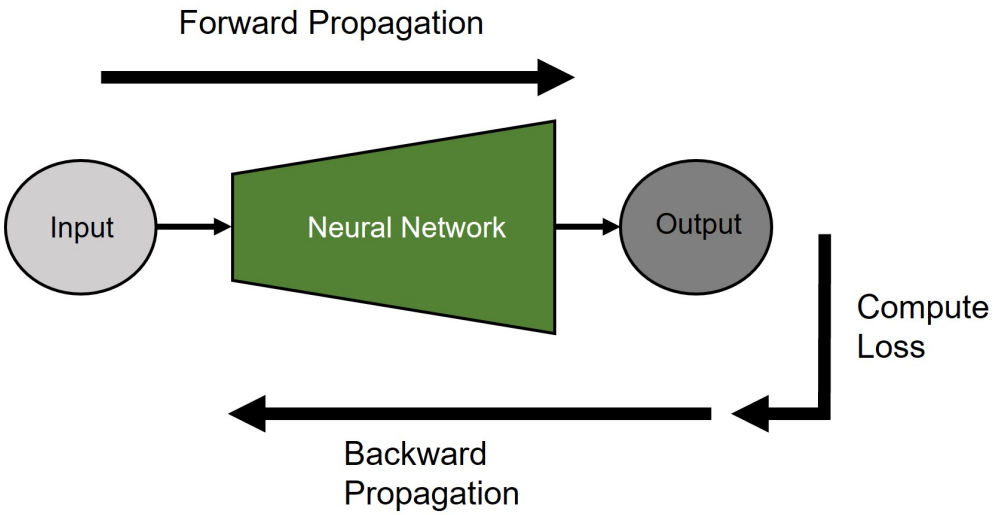


Figure 2.7: In model training, the input is fed to the neural network, and the loss is calculated from the output. Then, the backward propagation updates trainable parameters in the neural network.

2.2 Object detection

Object detection consists of object classification and object localization. The goal is to classify objects in the input image and give objects' local position in bounding boxes, as shown in Figure 2.8. Object detection can be used in many areas, such as autonomous driving, robot vision, or activity recognition. Convolutional Neural Networks have become the popular choice for model developers to improve both accuracy and speed. The challenges of designing an object detector are the limitation of computing power and the accuracy of prediction. The detector is expected to return local location and category of objects with optimized resource usage at high speed.

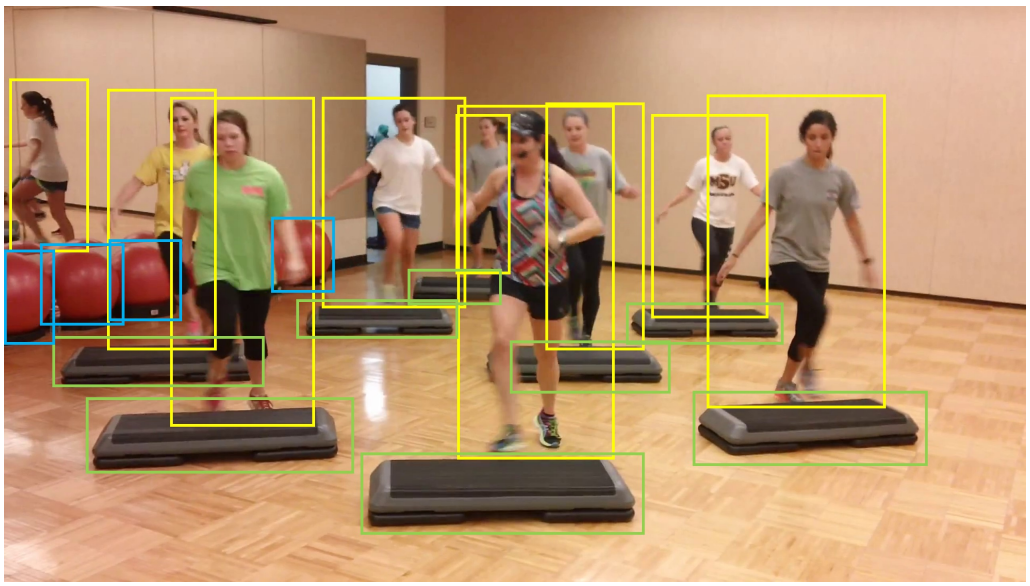


Figure 2.8: Object detectors detect objects in the frame. For example, this frame has humans, workout equipment, and balls in yellow, green, and blue.

One of the most critical parts of object detector training is the dataset. Each dataset is collected for a different purpose and method. A high amount of training images leads to high model accuracy. There are several public datasets available for object detection training, such as MS COCO [18], Open Images [19], or ImageNet [20]. There are two main categories of object detectors which are one-stage detector and two-stage detector. Definitions and examples are explained below.

2.2.1 One-stage object detector

The one-stage detectors are known as fast detectors. Their algorithm works simply as a regression problem. From the input image, class probabilities and bounding box locations are predicted directly. Thus, only one neural network is needed in this type of detector. One-stage detectors' accuracies are lower than most two-stage detectors, but the memory required and the processing times are also lower. Examples of the famous one-stage detector are YOLO [21] and RetinaNet [22].

2.2.1.1 YOLO

YOLO [21] stands for You Only Look Once. It treats object detection as a regression problem. A single convolutional neural network is connected to two Fully connected layers to do the classification and localization. The image is split into T by T grid cells. Each cell contains R anchor boxes where the anchor box is defined to handle each object class's scale and aspect ratio. The prediction as a bounding box is represented in 5 values: x, y, w, h, and p. The x and y are the coordinates of the bounding box's center in the local cell. W and H give information about the width and height of the bounding box. The confident score p represents the confidence of being specific class. Many versions of YOLO are developed to increase accuracy and speed, such as Tiny YOLO [23], YOLOv2 [24], YOLOv4 [25], or YOLOv5 [26]. YOLO stands for You Only Look Once. It treat object

2.2.1.2 RetinaNet

The Facebook AI research team develops RetinaNet [22]. The Focal loss function is introduced to replace the cross-entropy loss function as a new loss function. As a result, the detector reaches the accuracy of complex two-stage detectors with the speed of one-stage detectors.

2.2.2 Two-stage object detector

Two-stage detectors have two steps in the object detecting process. In the first step, the convolutional neural network, called Region Proposal Network, predicts regions of interest. The network decides that there is a high chance that objects are in the regions of interest. Next, another neural network takes the regions as input and gives predicted class and bounding box coordinates. This type of detector can perform at a very high accuracy but slow speed. Fast R-CNN [27] and Mask R-CNN [28] are the most successful two-stage detectors.

2.2.2.1 Fast R-CNN

The Fast R-CNN is the improved version of R-CNN [29]. First, the input image is fed to the convolutional neural network to generate the feature map. Then, the region of proposals is defined and reshaped from the feature map before sending into a fully connected layer. The output of the fully connected layer is called the feature vector, which is fed into the softmax layer and the fully connected layer in the next step. As a final prediction, the class and location are the output of the model.

2.2.2.2 Mask R-CNN

Facebook AI research team improve R-CNN [29] by adding the object mask feature. The Mask R-CNN can be seen as a segmentation tool. In addition to Fast R-CNN, the model has one more branch that predicts the regions of interest from the predicted bounding box.

2.3 Human pose estimation

Human pose estimation is the task that aims to estimate human body gestures and represent them as skeletons. Skeleton consists of joints which are parts of the body, such as shoulders, elbows, wrists, or hips. The link between every two joints is called limb. The example of a skeleton is shown in Figure 2.9. Developers design pose estimators for both two-dimension poses [30] and three-dimension poses [31]. This thesis focus on two-dimension human pose estimators.

Figure 2.9: Lines connect each body part. The lines, which are called limbs, are assembled into a skeleton. The skeleton represents the human pose and the positions of limbs, as shown in the figure.

The estimators can be used to recognize human activities, such as sports, sign language, training, driving, or being sick. Therefore, the applications are expected to be applied to various fields: Industry, Healthcare, Sports training, Communications, or the Film industry. The examples of human pose estimation applications are shown in Figure 2.10.

Figure 2.10: Human pose estimation can be helpful in various applications. For example, the gesture of a human playing sport can be analyzed to optimize the player's movement, as shown in the figure.

The resolution of images, nonvisible joints, occlusions, lousy lighting, or color of clothes are the difficulties of the human pose estimation task. The accuracy and speed are the main aims of each estimator. In the last decade, the majority of human pose estimators are based on the Deep learning technique. The convolutional neural networks are introduced to increase the efficiency of estimating human pose and decrease the operating time. Two-dimension human pose estimators are categorized into two main categories, which are Top-down approaches and Bottom-up approaches.

2.3.1 Top-down approaches

The top-down estimators start with detecting humans using object detectors. Once humans are filtered from detected objects, they are cropped into a batch of images. Next, the single image representing a single human is fed to the convolutional neural network to predict the location of each joint in this cropped image. The process keeps iterating until all detected humans pose are estimated. The local joint positions are projected into the original image coordinate in the last step. DeepPose [32], DensePose [33], and AlphaPose [34] are well-known top-down human pose estimators.

2.3.2 Bottom-up approaches

In contrast to top-down estimators, the bottom-up estimators predict all human joints in the frame at once. Then, they use different algorithms to assembly joints into humans. For example, Openpose [35] predicts Part Affinity Fields (PAF) in addition to joints. The PAFs represent predicted vectors pointing from joints to joints. Next, the vectors from PAFs are integrated to form human poses along with joints' locations. The popular bottom-up human pose estimators are [35] and DeepCut [36]. Figure 2.11 demonstrated the processes of the top-down approach and the bottom-up approach.

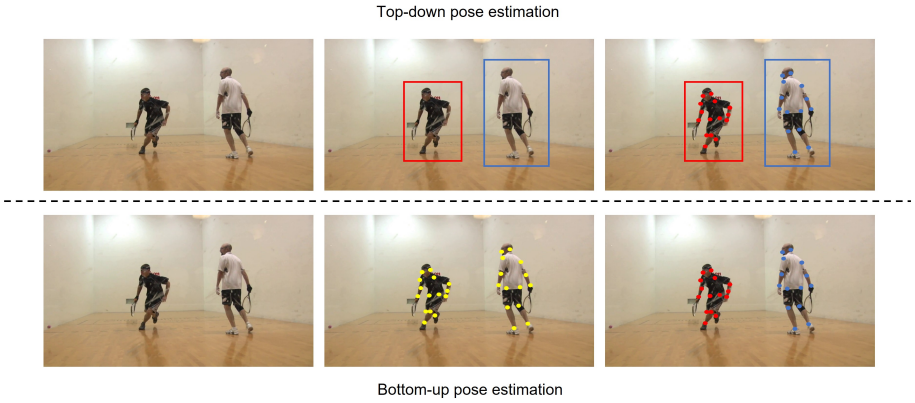


Figure 2.11: The top-down pose estimators detect humans and then estimate joint location for each human. In contrast, the bottom-up pose estimators detect every joint location in one process and assemble it into a human.

2.4 Multiple object tracking

Once object detectors detect objects, the locations and objects' categories are predicted. For some applications, locations and categories are not enough. The identity of each object must be assigned. In computer vision applications, the tracker aims to identify different objects and assign unique identification numbers to them across the frames. For example, The Car counter, which counts each category of cars passing one area each day, must not count the same car twice or more. Giving the identity to each detected car solves this problem. The counter counts the unique car, which appears in 2 frames, as one car. Figure 2.12 shows the comparison between the result from object detection and results from object tracking. Nowadays, there are many applications in which multiple object tracking plays an important role. People counting, football player tracking, and robot-path scheduling are famous examples of them.

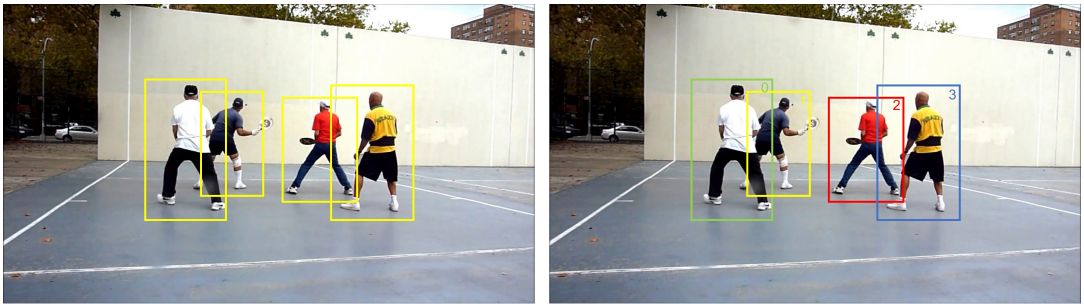


Figure 2.12: Object tracking aims to track objects across frames by assigning the unique identification number to objects. In comparison, Object detection focuses on detecting objects in each category without separating them.

The challenge of object tracking is similar to object detection. The variety of sizes and shapes of objects is solved by Anchor boxes [37], Multiple feature maps [38], and Feature pyramid network [39]. Feature engineering techniques, such as data augmentation, solve the class imbalance and limited data. In addition, various types of sorting algorithms reduce the speed of the identity assigning process.
There are two main types of multiple objects tracking methods, which are the online tracking method and the offline tracking method. The offline tracking approaches process a batch of frames to estimate human poses and assign unique IDs all at once. Online tracking approaches only exploit information, which is available until the current frame. A significant difference between the online and offline approaches is that the offline tracking approaches simultaneously analyze all frames, including the past, the current, and the future frames. On the contrary to its counterpart, the online approaches process only the current frame with past information to estimate human poses and assign identities. Both of them are measured and evaluated with the same metrics. Online tracking usually performs worst than the offline method in accuracy and speed. The offline method cannot be used in real-time applications due to the lack of future frames.

2.5 Edge computing

In the last decade, cloud computing [40] has been researched and implemented widely. Although implementing cloud computing means increasing the number of communication between edge devices and centralized computing units. Many applications require a real-time response at high frequency. The idea of increasing the computational power of edge devices and pushing some computing processes to the edge of the network is introduced as edge computing [41]. Edge computing aims to perform computations on edge devices such as mobile phones, cameras, or microprocessors. The illustration of edge computing is shown in Figure 2.13.

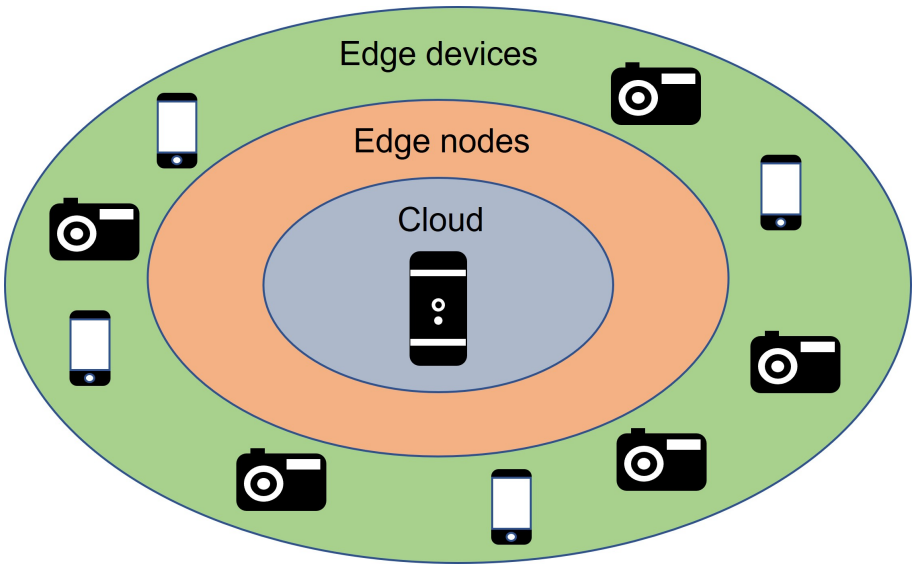


Figure 2.13: The cloud or server is the center of this communication. The connection between cloud and edge devices is provided by edge nodes, for example, routers. The user devices, such as cameras, minicomputers, or mobile phones, are called edge devices.

Small and less power-consuming computers are often the popular choice for edge devices, which are available on-site. Edge devices are low-performance devices compared to desktop computers or servers. Most of the researches are researched and developer to fit high-performance computer. Therefore, it is open research on edge devices.
The motivations of developing edge devices are described in [41]. To decentralized cloud is the first motivation. Some applications, such as geographically related applications, need to compute data closer to their source. The computation must be done on the edge devices. The complex computations frequently perform on the server regarding hardware limitations of edge devices. The second motivation is to perform a basic computation or filtering on edge devices before sending only necessary data to the server. Energy consumption is the main point of the third motivation. Cloud server uses a large amount of energy, and the number keeps growing over time. Decreasing data collecting and data transferring is the fourth motivation. According to the increasing cloud usage, a high amount of data is transferred at a

high frequency, and there is a high chance of data explosion. Edge computing helps decrease transferring unnecessary data and collecting a large volume of data in the data center. The last motivation is to develop advanced computation techniques. For example, the computation can be distributed to edge devices, edge nodes, or servers regarding their capabilities. The edge nodes are the devices that handle network traffic, such as routers, switches, or base stations.
Edge computing is still in the stage of improving. Several main challenges should be addressed in order to be at the same level as cloud computing. These challenges mean that there are many opportunities for academic research as well. A broad range of frameworks and programming languages must be handled to deploy applications on different edge devices. The lightweight libraries and algorithms are rapidly developed and researched. The available libraries are designed to work on computers and servers. Some of them are not available on edge devices or are not running at their optimal performance. So, many of their developers release the lighter version of their framework. For example, TensorFlow [42], a machine learning library from Google, is too heavy to run on mobile phones. So, the Google team introduce TensorFlow Lite to serve edge computing purposes.
Edge computing is prevalent in many areas. Autonomous vehicles, which is the upcoming trend, are an example of edge computing. Communicating with servers and other vehicles while moving can be a big problem. Each vehicle contains its computation power to handle data from its sensors. In many factories, machines require maintenance planning to operate smoothly. The maintenance date can be predicted using sensors in the machine. These data are sent to the data center, and the date is predicted on the server. Edge computing brings the calculation to the edge. The machine predicts its own maintenance time and sends the prediction to store in the server. Hospitals start implementing edge computing too. The patient monitoring is improved by introducing a new smart bed. The bed measures and monitors the patient's status. The unnecessary high-value communication system can be replaced with an uncomplicated system. In addition, reducing the large amount of data storing in the third-party cloud increases the system's security.

3
Methods

3.1 Training data

In this section, the data that is used to train models will be discussed. The section is divided into two parts: Datasets and Preprocessing data. The datasets section is for information about two public datasets, which are available to download, and the dataset collected from the edge computing environment for this thesis.

3.1.1 Datasets

Three datasets are using in this thesis. The Posetrack dataset and the COCO dataset are used to train and validate models performance. Finally, our new dataset is used for benchmarking models. The details of each dataset are described below.

3.1.1.1 PoseTrack dataset

The Posetrack dataset is collected to train models for the Posetrack challenge [43]. The challenge aims to benchmark video-based human pose estimation and human pose tracking. There are over 46000 frames and over 276000 human poses in the dataset. The size of the dataset is over 80 Gigabytes. Thus, the dataset is the largest dataset for human pose estimation and tracking. Frames are split into numerous video sequences. Each video contains humans acting on various activities to show body movement. In every frame, humans are presented at different scales. In some sequences, some people disappear, appear partially, or re-appear. Each video can have a length from 40 to 150 frames.

The key information in the dataset are human joints' locations, identity numbers, human locations, and ignore-regions. The ignore-regions are not used in this thesis. The human locations state the local bounding box of each human in order to crop the image on a single human. The identity number is given to each human who enters the frame until that person leaves the frame. Human joints' location is the annotation of 17 body parts as shown along with joint identity number in Table 3.1. The annotated body parts in the dataset are head, ears, nose, neck, shoulders, elbows, wrists, hips, knees, and ankles. Figure 3.1 visualize the skeleton, which is built from annotated joints.

The annotation for each human is stored in JSON format contains the following information:

- "bbox_head": A list of four elements giving the head location in the bounding box format.
- "keypoints": A list of fifty-one elements representing seventeen joints' locations. There are three numbers for each joint: x, y, and v. The x and y is the location of each joint. The v is one if that joint appears in the frame and 0 otherwise.
- "track_id": The unique identification number for each human.
- "image_id": A number telling the frame that this human appears.
- "bbox": A list of four elements giving the location of this human. The format for the bounding box is (x,y,w,h). The x and y locate the top left corner of the human's bounding box. The w and h are the width and height of the bounding box, respectively.
- "category_id": This number is always 1, stating that the category is human.

Joint ID	Joint name
0	Nose
1	Neck
2	Head
3	Left ear
4	Right ear
5	Left shoulder
6	Right shoulder
7	Left elbow
8	Right elbow
9	Left wrist
10	Right wrist
11	Left hip
12	Right hip
13	Left knee
14	Right knee
15	Left ankle
16	Right ankle

Table 3.1: The table shows 17 annotated body parts with unique identification numbers in Posetrack dataset.

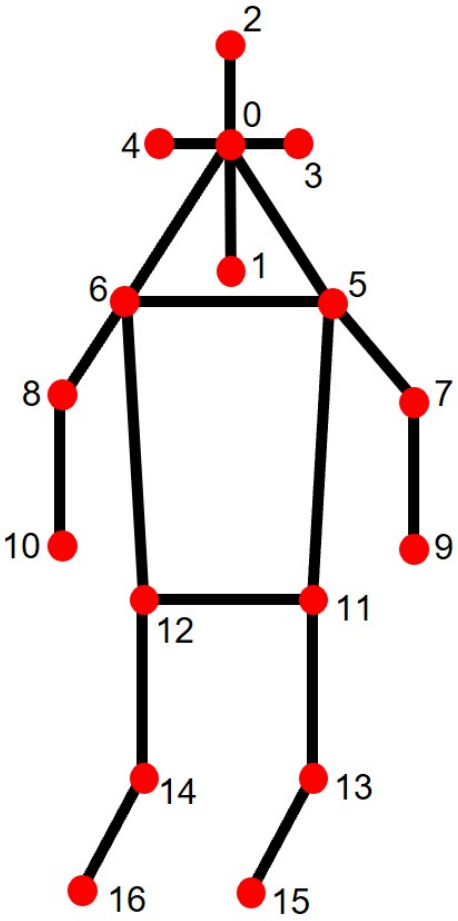


Figure 3.1: The skeleton illustrates 17 annotated body parts with unique identification number in Posetrack dataset.

3.1.1.2 COCO dataset

The Common Object in Context (COCO) dataset [18] is the most popular image dataset. The dataset has over 1.5 million objects in 80 categories for several purposes, such as object detection, instance segmentation, or human pose estimation. In this thesis, the human joints' locations are used to train the pose estimator. There are over 200000 frames and over 250000 human poses labeled in the dataset. The significant differences from the Posetrack dataset are that the COCO dataset cannot train the pose tracker because images are not in sequence and human body parts are labeled differently. The unique identification numbers (track ids) are not assigned to any poses because poses in each frame are not related. According to the difference in human joints' location annotation, the head and neck are not annotated in the dataset, but the left eye and right eye are added instead, as shown in Table 3.2 and Figure 3.2. The bounding boxes are annotated in the same way as in the Posetrack dataset.

Joint ID	Joint name
0	Nose
1	Left eye
2	Right eye
3	Left ear
4	Right ear
5	Left shoulder
6	Right shoulder
7	Left elbow
8	Right elbow
9	Left wrist
10	Right wrist
11	Left hip
12	Right hip
13	Left knee
14	Right knee
15	Left ankle
16	Right ankle

Table 3.2: The table shows 17 annotated body parts with unique identification numbers in COCO dataset. The head and neck are not annotated while the left eye and right eye are added to annotated body parts.

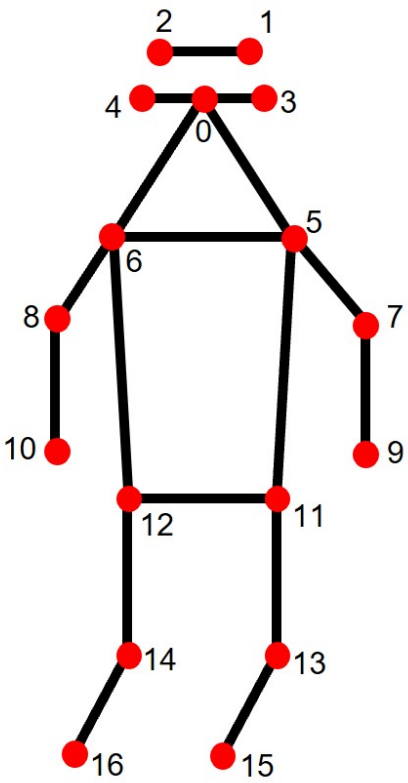


Figure 3.2: The skeleton illustrates 17 annotated body parts with unique identification number in COCO dataset.

3.1.1.3 Our new dataset

In order to benchmark models in edge computing, our new dataset is collected with the edge computing environment. Edge computing is usually deployed on a small computer with low computation power. For example, in computer vision applications, the cameras on edge devices always have low resolution. This thesis employs the Nvidia Jetson Nano [44] as the edge device. For the camera, Logitech C270 [45] is the HD USB camera widely used in many applications due to its price and quality. There are two types of situations in this dataset. The first situation illustrates various movements of a single human, such as walking, sitting, or talking on the phone. Two videos with 1000 frames contain 974 human poses. The example of an image in the first situation is shown in Figure 3.3.

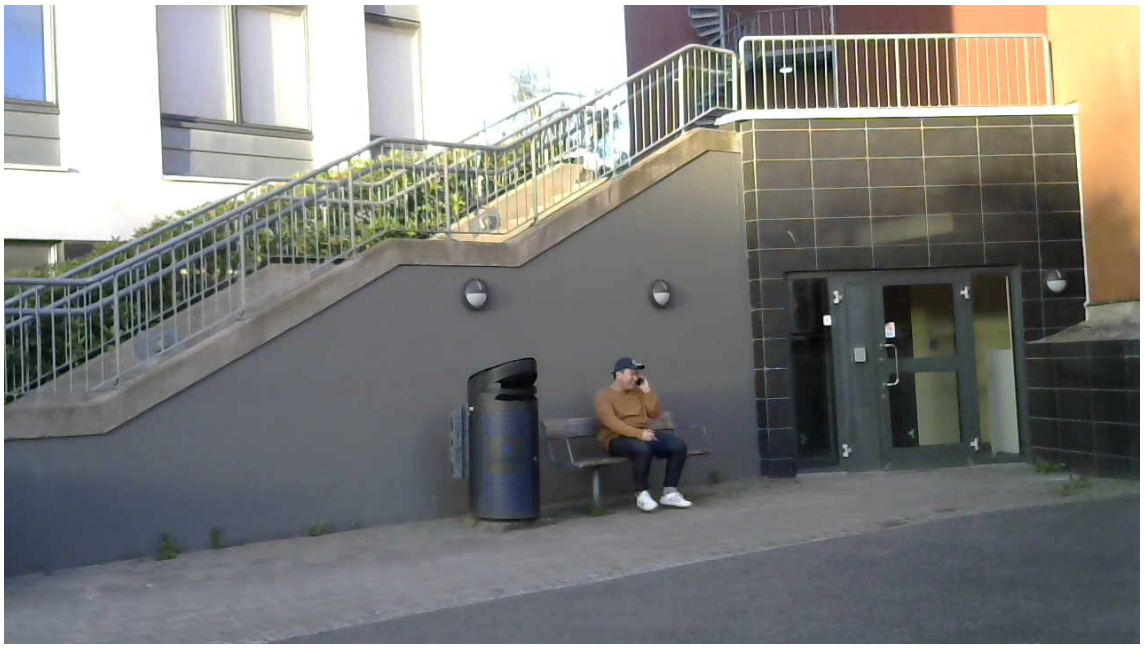


Figure 3.3: The single-human video contains a single human in different gestures in our new dataset. The example of the frame in the single-human video is shown in this figure.

Multiple human tracking is benchmarked in the second situation. Two separated videos capture 4 to 5 people moving and doing movements, as illustrated in Figure 3.4. The challenges of the second task are that some people or body parts are occluded partially in many frames, and people walk past each other causes difficulty to the tracking algorithm. There are 500 frames in 2 videos with 2250 human poses for this second situation. Table 3.3 shows the detail, such as the number of people, of each video. 3.1.2.

Situation	Video	No. of frames	No. of people	No. of poses
Single person	1	616	1	613
Single person	2	384	1	361
Multiple person	1	250	4	1000
Multiple person	2	250	5	1250

Table 3.3: The table shows number of frames, number of people, and number of human poses in each video of our new dataset. The videos are separated into 2 situations, which are single person tracking and multiple person tracking. Each situations contains 2 videos.

Figure 3.4: There are two multi-person videos in our new dataset. The first one consists of 4 people, while the second one has five people. In addition to the single-human video, the tracking task is added for benchmarking. The example of frame in the multi-person videos is shown in this figure.

Since the dataset is aimed for benchmarking, unique identification numbers and human joints' locations are annotated. Different from the Posetrack dataset and the COCO dataset, there are 15 annotated human body parts, as shown in Table 3.4. Four body parts are not included in the annotation to synchronize three datasets. The body parts are the head, neck, left eye, and right eye. The reason is that models are trained from the Posetrack and the COCO datasets, which have differences in these four body parts. The data preprocessing for the Posetrack dataset and the COCO dataset to match our new dataset is described in Section 3.1.2.

Joint ID	Joint name
0	Nose
1	Left ear
2	Right ear
3	Left shoulder
4	Right shoulder
5	Left elbow
6	Right elbow
7	Left wrist
8	Right wrist
9	Left hip
10	Right hip
11	Left knee
12	Right knee
13	Left ankle
14	Right ankle

Table 3.4: The table shows 15 annotated body parts with unique identification numbers in our new dataset. The number of annotated joints is different from the Posetrack dataset and the COCO dataset due to the simplicity.

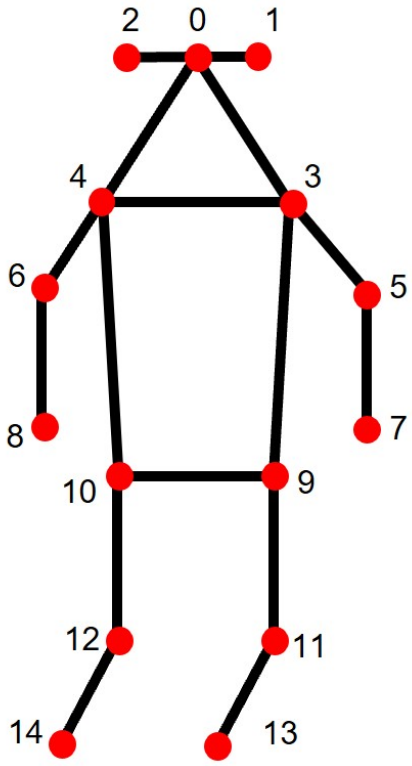


Figure 3.5: The skeleton illustrates 15 annotated body parts with unique identification number in our new dataset.

3.1.2 Preprocessing data

The data preprocessing is required to improve and match the training process. There are two processes in this thesis. First, the annotation reduction has been made to match the labeled human joints of three datasets. Second, in order to train pose estimators, human joints' location in the term of (x,y,v) is converted into a heatmap. The parameter v states the existence of a joint in a frame. The benefits and processes are explained in Section 3.1.2.1 and Section 3.1.2.2.

3.1.2.1 Annotation reduction

The Posetrack dataset and the COCO dataset contain 17 annotated human body parts. However, the trained models from the Posetrack dataset are not compatible with the COCO dataset. The reason is that there is a difference in the process of annotating. According to Table 3.1 and Table 3.2, joints one and two of the Posetrack dataset are the neck and the head, but they are the left eye and the right eye in the COCO dataset. Once models from the Posetrack dataset are benchmarked on the COCO dataset, the models try to predict head and neck instead of the left eye and right eye. This thesis aims to train models on both datasets to have a large amount of data. Then, those models are evaluated on our new dataset. Thus, joints one and two from both training datasets are filtered out before model training. Fifteen body parts represent humans in this thesis, as shown in Table 3.4 and Figure 3.5. Figure 3.6 illustrated fifteen body parts drawn on the image as a skeleton.

Figure 3.6: In this thesis, the training process and benchmarking process are based on 15 annotated body parts. Therefore, the 17 annotated body parts in the Posetrack and the COCO datasets are reduced into 15 annotated body parts, as shown in this figure.

3.1.2.2 Heatmap rendering

Human pose estimators are trained to predict human joints' location. Instead of using locations as ground truths, the heatmaps representing those locations are introduced in this process as in [46]. From the Posetrack challenge [43], most of the state-of-the-art models use are heatmaps predicting models. The heatmaps can be called confidence maps because each pixel in the heatmaps states the probability of the joint occurring at the pixel. Figure 3.7 shows joints' locations on the human body and the rendered heatmaps after preprocessing.

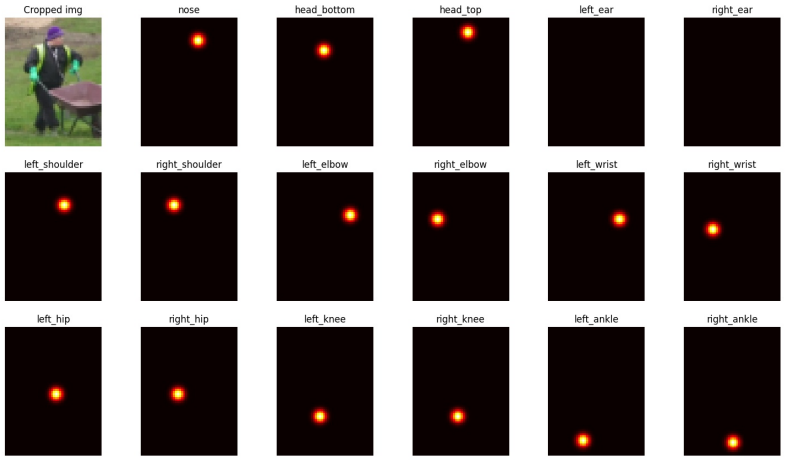


Figure 3.7: Each human joints' location is rendered into a heatmap by the gaussian equation as shown in Equation (3.1). The value in each pixel represents the probability that the joint appears in that pixel.

The process of encoding locations into heatmaps is called heatmap rendering. From 15 locations of a human, the rendered heatmap's shape is (HxWx15), where H and W are the output heatmap's height and width. For each body part, the position of the joint is represented as x_0 and y_0. Then, the empty matrix of size HxW is created. Each pixel of the empty matrix is represented by x and y, which are the pixel's coordinates. Finally, the value in each pixel is calculated by the gaussian equation as in Equation (3.1).

$$f(x,y) = e^{-(\frac{(x-x_0)^2}{2\sigma^2} - \frac{(y-y_0)^2}{2\sigma^2})} \tag{3.1}$$

σ is the value to be tuned, defining how the value spreads from the peak.

3.2 Models

There are two types of models that are researched in this thesis. The Flow Track (Section 3.2.1), the Light Track (Section 3.2.2), and the Face-to-pose Track (Section 3.2.4) are called top-down approaches, while the bottom-up approaches are represented by the Joint Flow (Section 3.2.3). Top-down approaches contain three steps: human detection, human pose estimation, and human tracking. For bottom-up approaches, model prediction, human assembly, and human tracking are their procedure. The Face-to-pose Track is developed for this thesis aiming to use in edge computing applications. The following sections describe the detail of each model.

3.2.1 Flow Track

The main purpose of Flow Track [47] is to be the simple baselines for human pose tracking algorithms. The model is developed to decrease the complexity of the pose tracking algorithm. The human detector, human pose estimator, and pose tracker are built separately as modules. The process starts with creating an empty list to store detected poses called Q. After Q is created, the process shown in Figure 3.8 repeats in a loop until the last image or receiving stop command. Firstly, the image is fed to the human detector module. The output of the human detector module is the cropped images of each person who appears in the image. The human pose estimator module takes the cropped images as input and gives the detected human's predicted poses as an output. The unique identification number (Track ID) is not assigned to any pose in this step. The pose tracker module compares these poses to the stored poses in Q to assign stored Track IDs to them. If some poses do not match poses in Q, the new Track ID will be assigned to the pose. At the end of a loop, the predicted poses with Track IDs are saved into Q.

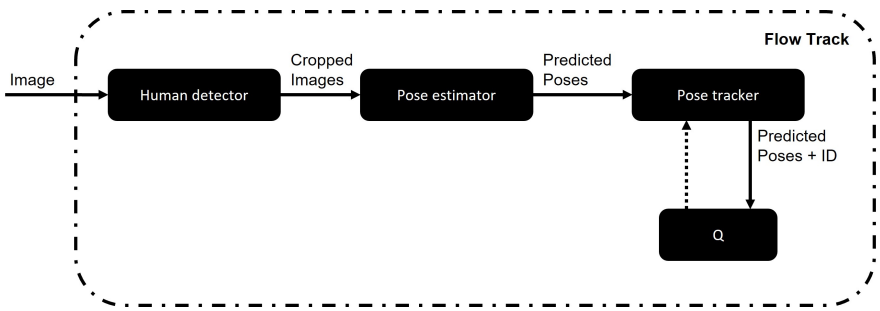


Figure 3.8: The flow, when each image is fed to the Flow Track, is shown in this figure. The Flow Track follows the top-down method. The Human detector plays an essential role in this model.

The human detector module consists of YOLOv4 [25] as a human detector and functions to crop images using bounding boxes detected by the detector. As mentioned in Section 2.2.1.1, YOLOv4 is the improved version of YOLO (You only look once). The core neural network, which is also called the backbone, is changed to CSPDarknet53 [25]. In addition, techniques, such as spatial pyramid pooling module or PANet path-aggregation neck, are added. YOLOv4's pre-train model is also

available publicly in many frameworks, including TensorFlow, Pytorch, TensorRT. The convolutional neural network (CNN) plays an essential role in the human pose estimator module. Before feeding to the CNN, the cropped image of a single human is resized to 256x192x3. Then, Resnet152 [48], the well-known feature extraction network, is connected to three transpose convolutional layers [49] and one convolutional layer with the same padding to predict heatmaps, as shown in Figure 3.9. The shape of the output from the CNN is 64x48x15. If the peak value is higher than the threshold, it is selected as human joints' location. For the heatmap whose value does not pass the threshold, the joint related to the heatmap is defined as a disappeared joint. Finally, the decoded values from heatmaps are stacked together as a list of human joints' locations. The list, which can be called a skeleton, represent human in the cropped image.

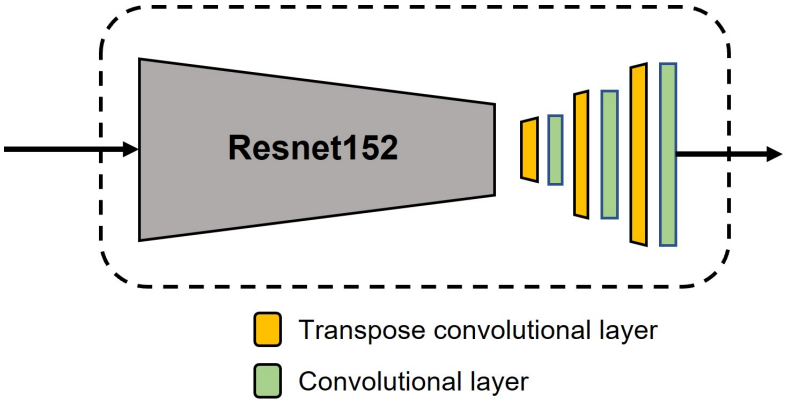


Figure 3.9: The architecture of Flow Track's human pose estimator module consists of the Resnet152 neural network, three transpose convolutional layers, and three convolutional layers.

The tracking module creates the similarity matrix to store the similarity of predicted poses and stored poses. For M predicted poses, the similarity matrix's size is MxN, where N is the number of stored poses in Q. One pose is drawn from the predicted poses to measure the similarity with another drawn pose from the stored poses. The average Object Keypoint Similarity (OKS) is selected to measure pose similarity in this thesis. The average OKS can be calculated as in Equation (3.2).

$$\overline{OKS} = \frac{\sum_{i=0}^{14} e^{(-\frac{{d_i}^2}{2s^2{k_i}^2})}}{\sum_{i=0}^{14} v_i} \tag{3.2}$$

The d_i is the euclidian distance between the predicted pose's keypoint and the stored pose's keypoint. The area of the bounding box is calculated by multiplying width and height and represented as s. In order to control fall off, constant k_i is defined. The value of k is different due to each body part, as shown in Table 3.5. The joint validity (v_i) is one, if the joint appears in the frame, and zero otherwise. After calculating similarities of ever poses, the greedy matching algorithm, shown in [50], is implemented to assign the unique identification number from the stored pose to a similar predicted pose. The new unique identification number is assigned if a predicted pose with a low similarity does not match any stored poses.

Joint	k_i
Nose	0.026
Left ear	0.035
Right ear	0.035
Left shoulder	0.079
Right shoulder	0.079
Left elbow	0.072
Right elbow	0.072
Left wrist	0.062
Right wrist	0.062
Left hip	0.107
Right hip	0.107
Left knee	0.087
Right knee	0.087
Left ankle	0.089
Right ankle	0.089

Table 3.5: The constant k is defined to control fall off. The value is depend on each joint.

3.2.2 Light Track

Light Track [51] aims to be a fast human pose tracking approach. The main idea of Light Track is to minimize the usage of the human detector module because the module requires a high amount of computation power. Therefore, the keyframes are images feeding to the human detector module. The non-keyframes are processed with another detection algorithm. First of all, the storage Q is defined as the same as in the Flow Track. In addition, the previous frames storage for storing tracked poses in the previous frame is created as an empty list called PQ. Then, as illustrated in Figure 3.10, the repeating procedure keeps running until it stops or running out of images. The process starts with deciding if the current image is a keyframe or not. The duration between each keyframe is called K. K is the value to be tuned by the user. For example, suppose K is set as three. In that case, the keyframes are frame 0, frame 3, frame 6, and multiples of K. If the image is keyframe, it is fed to the human detector module, human pose estimator module, and human tracker module, respectively. The human detector module and the human pose estimator module work the same way as the Flow Track. The pose tracker module compares predicted poses with poses from PQ instead of Q. At the final step, the pose tracker module's output, the list of poses with Track IDs, is added to Q and replaced poses in PQ as a new PQ. For non-keyframes, the bounding boxes of the previous frame's poses in PQ are used to detect a human in the current frame. The bounding boxes are expanded horizontally and vertically for 10% of the original sizes to compensate if humans move from the last frame. Images are cropped out from the current frame using the expanded bounding box before sending into the human pose estimator module. There is a function in the human pose estimator module to detect if people disappear from the cropped image to handle ghost detection.

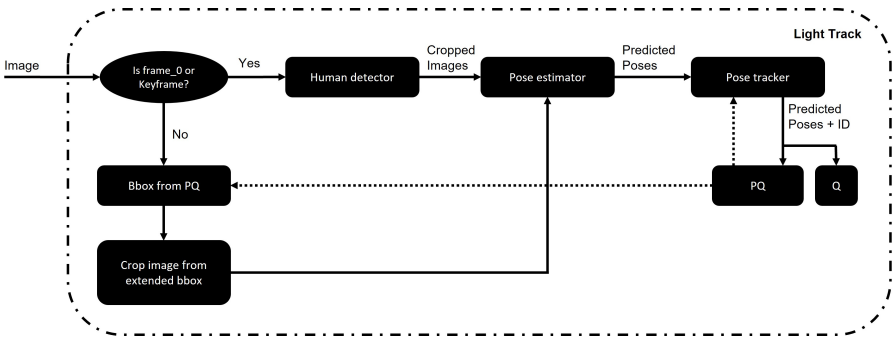


Figure 3.10: In addition to Flow Track, the flow of the Light Track get the boundary box from the previously stored data in non-key frames. In the keyframes, the process is similar to the Flow Track.

The human detector module is the same as in Flow Track. The YOLOv4 is selected to be the human detector, which is only used with keyframes. The human pose estimator module is very similar to the human detector module of Flow Track. The only exception is that the input image's size to CNN is (384,288,3) according to the original setting in [51]. The developers believe that the large input image leads to higher accuracy in the pose estimation task.

The similarity in Light Track is calculated from the bounding box instead of the human pose. Once a predicted pose is drawn from all poses to measure similarity with a stored pose from storage PQ, the Intersection over Union (IoU) of bounding boxes is computed. As illustrated in Figure 3.11, the IoU can be calculated by dividing the overlap area between two bounding boxes by the area of the union of two bounding boxes. After the similarity matrix is filled, the unique identification number assignment is the same as the Flow Track.

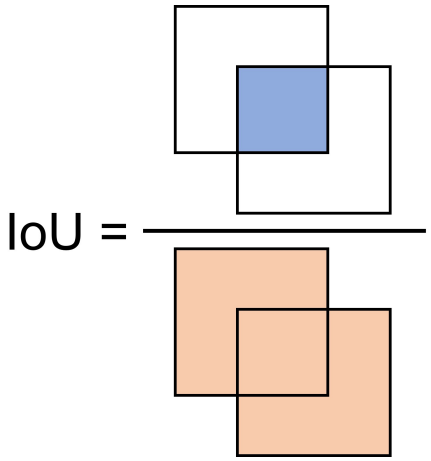


Figure 3.11: The Intersection over Union (IoU) is calculated by the intersection of two bounding boxes divided by the union of the bounding boxes, as illustrated in the figure.

3.2.3 Joint Flow

The Joint Flow [52] is one of the bottom-up pose tracking approaches. The bottom-up approaches aim to predict human joints' location and relationship map in one step. Then, the human skeleton is formed in the next step. Finally, the relationship between current-frame poses and previous-frame poses is extracted from the predicted relationship map. Before explaining the model, the limb concept is defined. The limb is the connection between two human body parts. There are 16 limbs in this thesis. Table 3.6 shows the name, limb's number, and the body part connected by the limb.

Limb ID	Limb name	Connected parts
0	Left cheek	Nose, Left ear
1	Right cheek	Right ear, Nose
2	Left neck	Nose, Left shoulder
3	Right neck	Nose, Right shoulder
4	Left upper arm	Left shoulder, Left elbow
5	Right upper arm	Right shoulder, Right elbow
6	Left lower arm	Left elbow, Left wrist
7	Right lower arm	Right elbow, Right wrist
8	Shoulder	Right shoulder, Left shoulder
9	Left body	Left shoulder, Left hip
10	Right body	Right shoulder, Right hip
11	Bottom	Right hip, Left hip
12	Left upper leg	Left hip, Left knee
13	Right upper leg	Right hip, Right knee
14	Left lower leg	Left knee, Left ankle
15	Right lower leg	Right knee, Right ankle

Table 3.6: The table shows details of 16 limbs using in this thesis. Limb is the path connecting 2 different body parts.The skeleton from these limbs is shown in Figure 3.5

This model has three storages: the pose storage Q, the previous pose storage PQ, and the previous frame PI. Q and PQ are created as empty lists. The PI is defined as the empty frame which is used as a memory for one previous frame. When the first frame enters the model, the frame is saved to PI to predict the next frame and fed to the Openpose module, which is explained in the next paragraph. The output of the Openpose module is the predicted human poses in the first frame. Then, these poses are stored in the Q and PQ. If an image is not the first frame, the image is fed into the Joint Flow module together with PI and PQ. Then, the predicted poses, which is the output from the module, are added into Q and saved as a new PQ. The PI is updated to be the current frame before the new frame arrives. This process keeps running until there is no frame left or the stop command is sent. The flow of the process is shown in Figure 3.12.

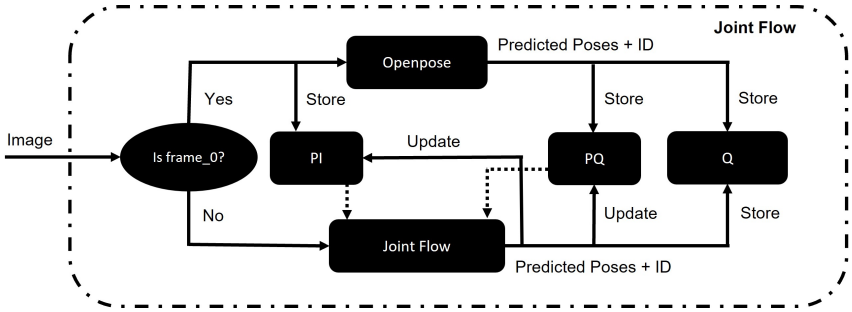


Figure 3.12: The bottom-up pose tracking is more complicated than top-down pose tracking. The aim is to estimate all human body parts in the frame simultaneously before assembly into a human. The relationship between two frames is also predicted as TFF to link detected humans to the stored information from the previous frame.

The Openpose module consists of the CNN and the skeleton assembly function. The CNN used in this module is called the Openpose model [35]. The Openpose model takes images as input. The model's outputs are the Belief Map (BM) and the Part Affinity Field (PAF). The BM is the heatmap telling the confident score of joint appearing in each pixel. The PAF is the matrix that shows vectors representing limbs. The human joints' locations are extracted from the BM by detecting the peaks. Figure 3.13 shows the example of the Belief Map and the Part Affinity Field.

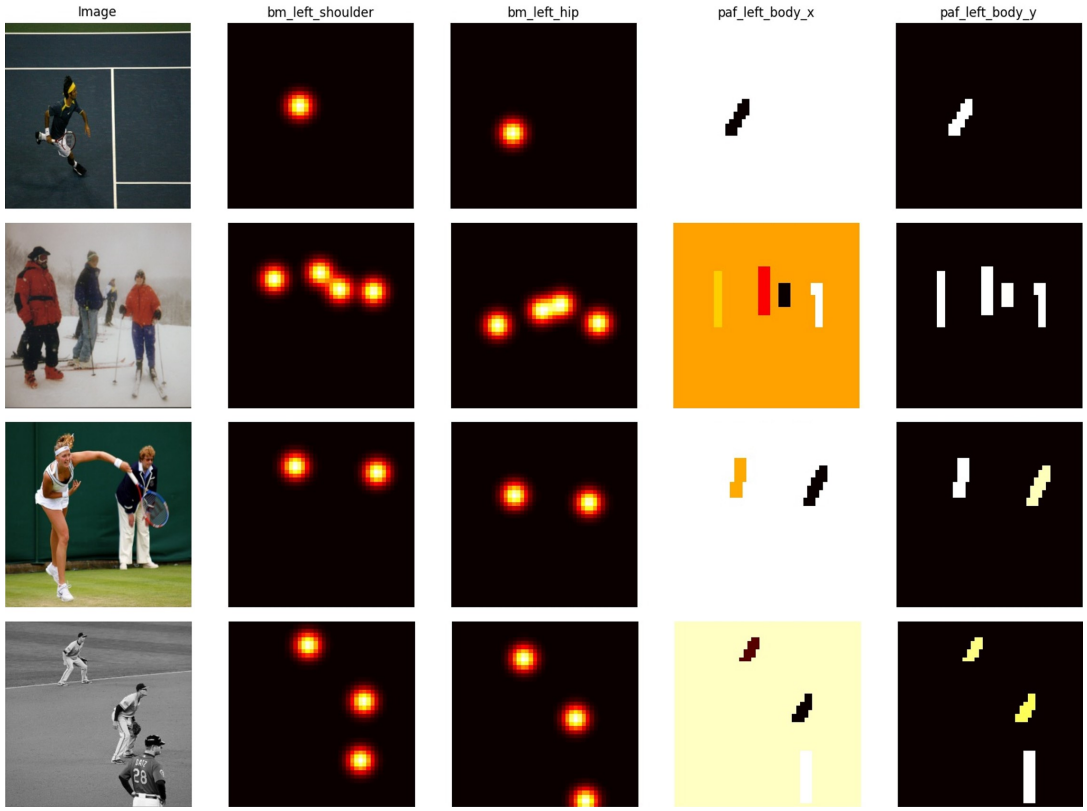


Figure 3.13: The Belief Map (BM) is the same as the heat map in the top-down pose tracking. At the same time, the Part Affinity Field (PAF) gives the value of the unit vector representing the limb.

One heatmap might contain more than one peak because the Openpose model predicts every person in the frame in one shot. The network is initialized with the first ten layers of VGG-19 [53] to get the feature maps (F). Then, the stage is defined as a set of neural networks. There are two networks in each stage called branches. The branch in the first stage contains three convolutional layers with a three-by-three kernel and two convolutional layers with a one-by-one kernel. Then, the outputs from branch one and branch two are concatenated together with original feature maps. After the first stage, every stage has five convolutional layers with a seven-by-seven kernel and two convolutional layers with a one-by-one kernel. The outputs of each stage, the BM and the PAF are concatenated with original feature maps as in the first stage. There are six stages in this thesis, according to [35]. The flow of data and the architecture of the Openpose model are illustrated in Figure 3.14.

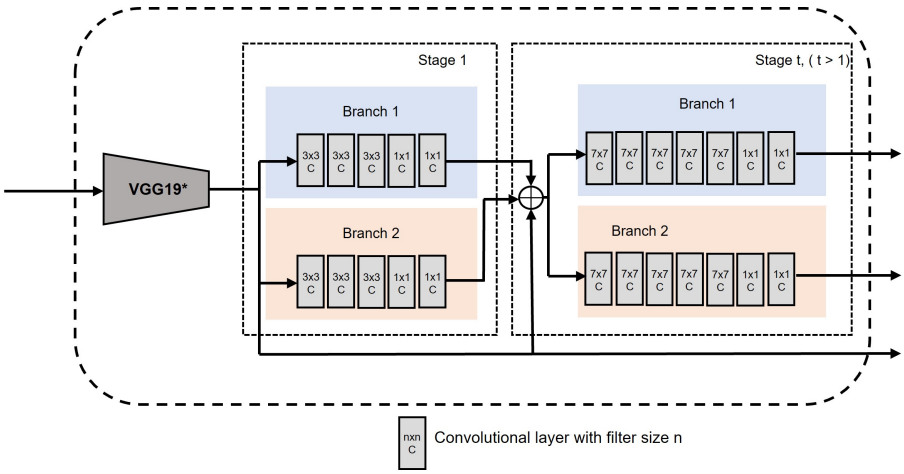


Figure 3.14: The architecture of the Openpose model consists of many stages. The first stage is unique. The stages, which come after the first stage, are repeated. The outputs from the last stage are the Belief Map (BM) and the Part Affinity Field (PAF).

The skeleton assembly function starts with detecting human joints' locations. Then, the related joints are selected for each limb to calculate if it can be formed as the limb. Finally, the score (E) to decide if two joints are the limb members can be calculated from Equation (3.3).

$$E = \int_{u=0}^{u=1} L_c(p(u)) \cdot \frac{d_{j2} - d_{j1}}{||d_{j2} - d_{j1}||_2} \tag{3.3}$$

$L_c(p(u))$ is the value from PAF in the term of the unit vector. d_{j1} and d_{j2} are the joint's locations of both candidates. The $p(u)$ interpolates the position of the two body parts d_{j1} and d_{j2}. The $p(u)$ can be computed from Equation (3.4). In practice, the value of u is sampled with the constant space. After the step, limbs are matched into the human poses using a matching algorithm [35].

$$p(u) = (1 - u)d_{j1} + ud_{j2} \tag{3.4}$$

For the non-first frames, the Joint Flow module[52] processes the current frame together with the previous frame stored in PI. Inside the module are the Joint

Flow model, the skeleton assembly function, and the tracking function. The Joint Flow model is the extended version of the Openpose model. There are two parallel Openpose models which take two images as inputs. The outputs from the parallel models are two features maps (Fs), two BMs, and two PAFs. Next, the features maps and the BMs are fed to the additional network. This additional network contains five convolutional layers with a seven-by-seven kernel and two convolutional layers with a one-by-one kernel. The result of the additional network is the Temporal Flow Field (TFF) [52]. The TFF states vectors showing how each body part moves from the previous frame. The diagram of the Joint Flow model is illustrated in Figure 3.15. In this step, the outputs from the Joint Flow model are the current frame's BM, the current frame's PAF, and the TFF.

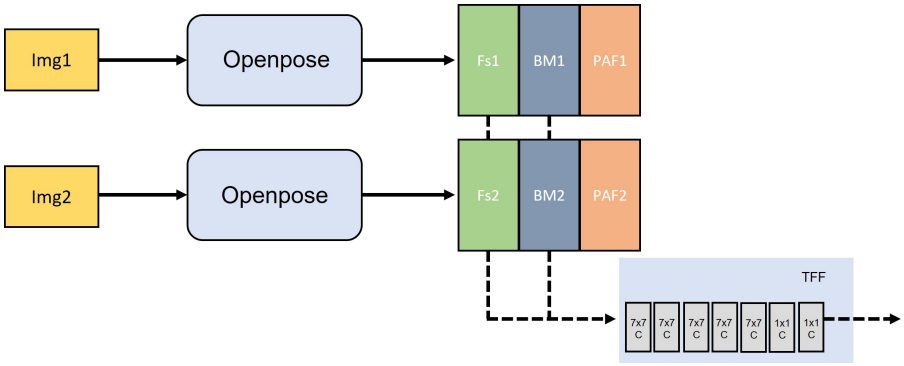


Figure 3.15: The Joint Flow model consists of two parallel Openpose models. The outputs from the models are concatenated before feeding to the additional network. The additional network predicts the relationship between 2 input frames.

The skeleton assembly function works the same way as in the first frame to get the predicted human poses. Then, the tracking function calculates the relativity of the predicted human poses and the previous human poses from PQ using TFF. The relationship between each pair of poses is calculated with Equation (3.5). The Tc is the value from TFF, and other parameters are the same as in Equation (3.3). The approach is very similar to the skeleton assembly function. The greedy matching is implemented to assign the old identification number to poses with a high relationship score. The new identification numbers are assigned to the poses that do not match any poses from previous frames.

$$E = \int_{u=0}^{u=1} T_c(p(u)) \cdot \frac{d_{j2} - d_{j1}}{||d_{j2} - d_{j1}||_2} \tag{3.5}$$

3.2.4 Face-to-pose Track

The Face-to-pose Track is developed for this thesis aiming to use in edge computing applications. The idea of the Face-to-pose Track is to use the Deep learning technique to predict the similarity instead of the calculation as in the Flow Track and the Light Track. The structure of the Face-to-pose Track is similar to the Flow Track. There are three modules: the human detector module, the human pose estimator module, and the pose tracker module. The human detector and pose estimator modules are the same as the Flow Track, both in architecture and setting. In the tracking module, there are the siamese model and the identity assigning function. The overview of the Face-to-pose Track approach is illustrated in Figure 3.16.

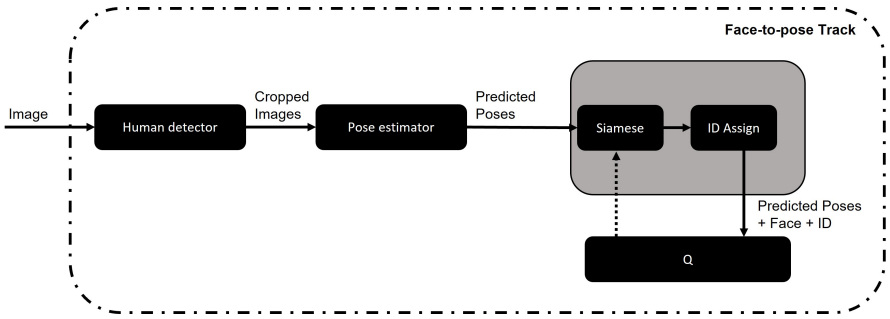


Figure 3.16: The Face-to-pose Track's concept is similar to the Flow Track. However, face recognition is implemented to predict the similarity instead of pose similarity calculation in the Flow Track.

In the first step, the faces of each candidate are cropped from the predicted poses in the human pose estimation module. The location of the nose is checked. If the nose is not detected in the pose, the ears are checked in the next step. The pose is marked as "no head" if both ears are not found. The siamese model consists of two parallel neural networks aiming to extract features from the input. The reason for using a neural network to predict the similarity of the two poses is that the neural network can be deployed on the TensorRT framework [54]. The framework operates on the graphic card, which is faster than in the CPU. In this thesis's hypothesis, operating neural networks on TensorRT can be faster than matrix operation and nested loop programming in the CPU. This Deep learning technique is called Siamese facial recognition [55]. Each network in the siamese model, called the features extractor, starts with three convolutional layers and three max-pooling layers at the end of each convolutional layer. Then, the output from the last max-pooling layer is fed to the global-average-pooling layer and two fully connected layers. The output from the features extractor is the array with 128 slots called the feature array. The siamese model uses the features extractor to extract features from two faces of candidates. Finally, the euclidean distance of two feature arrays is calculated before using the sigmoid function[Equation (3.6)] to let the score stay in range zero to one. The structure of the siamese model is shown in Figure 3.17.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{3.6}$$

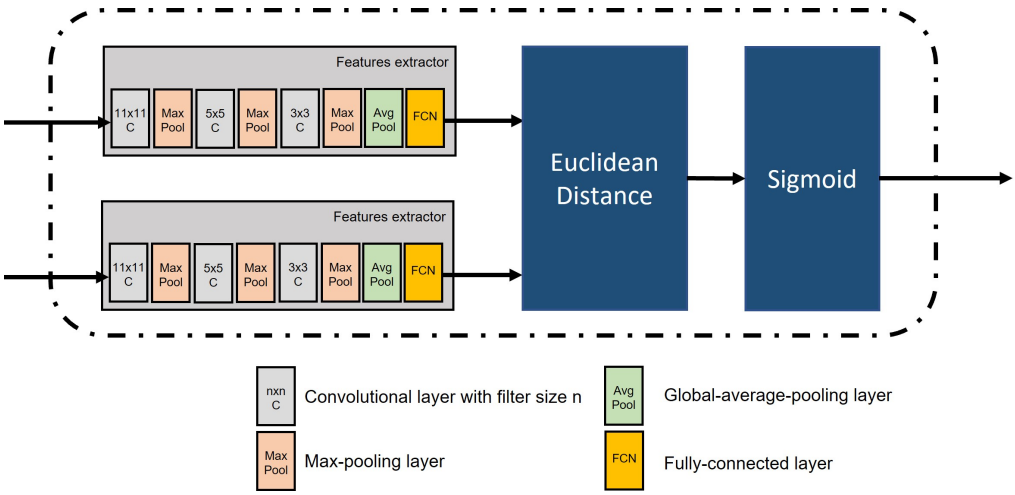


Figure 3.17: Face recognition is made of the siamese model. The siamese model or twin model is the concept of using two identical networks to extract features from two inputs. Then, the comparison of the features is turned into a similarity score.

After the scores between each pair of poses are calculated, the identity assigning function uses a greedy matching algorithm to assign an identification number to each predicted pose. Then, the poses are stored in the storage Q.

3.3 Training setting

In the training process, models are built on TensorFlow 2 framework [42] with Python. The TensorFlow 2 is the open-source platform for Machine learning and Deep learning developed by Google. The training script is written on Google colab. Google Colab is the online virtual machine that allows users to write and execute Python's script in a notebook. The advantage of using Google Colab with TensorFlow 2 is that the environments and the libraries needed to train the model are pre-installed in the virtual machine. In addition, the powerful graphic card, which shortens the training process, is available for free. In this section, the training parameters or training environments for each model are described. There are two types of models used in the top-down approaches: the human detector model and the human pose estimator model. The bottom-up approach contains one primary model and another extended version of the main primary, as Section 3.2.3 explains. The trained YOLOv4, described in Section 3.3.1, is selected as a human detector model for Flow Track, Light Track, and Face-to-pose Track.

3.3.1 Pre-trained human detector

The concept of the pre-trained model is that the parameters of the trained model are saved and shared with the open-source community. The pre-trained model can be used directly or can be trained to do similar tasks. In this thesis, the human detector model is the pre-trained YOLOv4 from [56]. This YOLOv4 is working on the TensorRT framework, which is discussed in Section 3.4.2. The original model

is designed to detect objects in many categories. The modification is made by connecting the original model with the filter function. This filter function select only human detected by the model.

3.3.2 Flow Track's parameters

The human pose estimator in Flow Track is trained with the PoseTrack dataset and the COCO dataset. There are several parameters to tune to this training. Each parameter is explained in this section. First, batch size is the number of images feeding to the model in one step of training. Second, when the entire dataset is fed to training, the training unit is called one epoch. The learning rate indicated how quickly the model learns per one batch. Optimizer is the algorithm used to maximize the efficiency of the training. There are several optimizers built in to TensorFlow 2 framework, such as Adam [16], RMSProp [17], or SGD [57]. The early stopping is the Deep learning's technique to stop the training if the score of the validation dataset is not improving anymore. The technique aims to prevent the model from working well only on the training dataset or overfitting. In this technique, the parameter to be tuned is called early stop patience or the patience. The patience tells how many epochs to wait if the score is not improving before stop the training. The tuned parameters for Flow Track's human pose estimator model training are shown in Table 3.7.

Parameter's name	value
Batch size	64
Number of epoch	60
Learning rate	0.0001
Optimizer	Adam
Early stop patience	6

Table 3.7: The parameters used for training human pose estimator model for both the Flow Track and the Face-to-pose Track.

3.3.3 Light Track's parameters

The human pose estimator model in Light Track is different from the Flow Track, but the training process and parameters to be tuned are the same. The parameters can be found in Table 3.8.

3.3.4 Joint Flow's parameters

There are two models to be trained in the Joint Flow: the Openpose model and the Joint Flow model. The Joint Flow model is the extension of the Openpose model. The trained Openpose model is plugged in the Joint Flow model directly. Thus, there is only the extended part to be trained in the Joint Flow model. Table 3.9 contains parameters for the Openpose model, and Table 3.10 contains parameters for the extended part in the Flow Track model.

Parameter's name	value
Batch size	16
Number of epoch	60
Learning rate	0.0001
Optimizer	Adam
Early stop patience	6

Table 3.8: The parameters used for training human pose estimator model for Light Track.

Parameter's name	value
Batch size	16
Number of epoch	30
Learning rate	0.0001
Optimizer	SGD
Early stop patience	6

Table 3.9: The parameters used for training the Openpose model.

3.3.5 Face-to-pose track's parameters

Since the human pose estimator model is the same as the Flow Track, the same training process and parameters are applied to the Face-to-pose track's model. Therefore, the parameters for the pose estimator, which is sharing with the Flow Track, are stated in Table 3.7. In addition, the siamese model for similarity prediction is another model to be trained. Therefore, the parameters for the siamese model training are in Table 3.11.

3.4 Deployment

The goal of this thesis is to evaluate models on edge computing. However, in order to deploy models on edge devices, there is one major challenge to overcome. The device in this thesis is the Nvidia Jetson Nano. The memory size of edge devices is always smaller than computers or servers. The model, built from some framework, cannot deploy on edge devices because the size is too large for the memory. Some can be installed to edge devices but cannot operate because the installation takes all the memory available. In this thesis, all models are created on TensorFlow 2 framework. TensorFlow 2 models cannot operate according to the size of the required available memory on the device is not enough. Thus, the TensorFlow 2 models are converted into the TensorRT model to deploy on Nvidia Jetson Nano. This section explains the selected edge device, the deployed framework, and the deployment process.

Parameter's name	value
Batch size	16
Number of epoch	60
Learning rate	0.00002
Optimizer	Adam
Early stop patience	6

Table 3.10: The parameters used for training the Joint Flow model.

Parameter's name	value
Batch size	16
Number of epoch	100
Learning rate	0.001
Optimizer	Adam
Early stop patience	6

Table 3.11: The parameters used for training the siamese model for the Face-to-pose Track.

3.4.1 Nvidia Jetson Nano

Nvidia Jetson Nano [44] is a compact size computer with mighty computation power developed by Nvidia. In addition, the Jetson Nano has a built-in graphics processing unit (GPU), allowing developers to design parallel computations. Working with machine learning and deep learning applications is the prominent point of the device. The operating system (OS) of the device is running on a MicroSD card. Thus, the scaling of production is done by cloning the card. Table 3.12 shows the specifications of the Nvidia Jetson Nano.

GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI and display port
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I2C, I2S, SPI, UART
Mechanical	69 mm x 45 mm, 260-pin edge connector

Table 3.12: The official specification of Nvidia Jetson Nano from the specification sheet.

3.4.2 TensorRT

Similar to TensorFlow 2 and Pytorch [58], Nvidia TensorRT [54] is a machine learning framework released by Nvidia. The disadvantage of TensorRT is that there is no training framework on it, which TensorFlow 2 and Pytorch have. The framework is compatible only with inference. The concept of TensorRT is the parallel programming model on the GPU. The deployed model on TensorRT is a lot smaller than in TensorFlow 2, which suits the small memory computer. The TensorRT can be used in C++ and Python, the popular programming language for machine learning applications. Operating with the parallel programming model, the model performs up to 40 times faster than the CPU inference [54].

3.4.3 Model Conversion

Since the TensorRT framework is not for training, the models must be trained in other frameworks. In this thesis, models are built and trained on TensorFlow 2 before deploying on TensorRT. The conversion from TensorFlow 2 model (TF model) to the TensorRT (TRT) model is not simple. According to the official documentation [59], there are two ways to deploy the TF model on TensorRT. The first way is to use the TF-TRT library to run the TF model directly on the Jetson Nano. In this case, the model's size is approximately the same as deploying the TF model on TensorFlow 2, and the inference is very slow because the memory of the device is almost full. In this thesis, the second way is implemented. The TF model is converted into the Open Neural Network Exchange (ONNX) model in the first step using a library called keras2onnx as mentioned in [59]. Then, the ONNX model is transferred to the Nvidia Jetson Nano before using the built-in script to convert the ONNX model to the TRT model. Finally, the TRT model can run on the TensorRT framework using the built-in library from Nvidia called tensorrt. The process of converting the TF model to the TRT model is summarized in Figure 3.18.

Figure 3.18: In order to deploy the TF model on the Nividia Jetson nano, The TF model is converted into ONNX using the open-source script called keras2onnx. Then, the ONNX model is adapted into the TRT format by the Nvidia built-in script in the TRT library.

3.5 Evaluation

3.5.1 Tasks

The models in this thesis are benchmarked in three tasks: Multi-frame person pose estimation, Multi-person pose tracking, and Live tracking. In the Multi-frame person pose estimation, the video sequences of humans in different activities are fed to the models. The task is to estimate human poses both in single-person videos and multi-person videos. The mean Average Precision (mAP) explained in Section 3.5.2 is used to score each model. In the second task, the goal of the Multi-person pose tracking is to track humans and estimate poses simultaneously. The task results are benchmarked with the Multiple Object Tracking Accuracy (MOTA) as explained in Section 3.5.3. Finally, Live tracking is the task that the device captures images and tracks poses simultaneously. In this task, the evaluation is done with evaluation metrics, which are mAP and MOTA metrics. The speed of each model is stated by the Frames Per Second (FPS). The FPS gives the number of frames that the model can process in one second.

3.5.2 Mean Average Precision (mAP)

The mean Average Precision [60] shows how accurate the model can predict human joints' locations. The calculation is done separately for each joint. Thus, there are the average precisions in every body part. Then, the mean of those average precisions is computed. The calculation process starts with the predicted points and the ground truth points for each body part and frame. Next, the Object Keypoint Similarity(OKS) between each pair of points is calculated. The equation of OKS is similar to Equation (3.2), but the score is not averaged in this case. Instead, the equation for calculating OKS is shown in Equation (3.7). The d is the distance between two points. The area of ground truth bounding box related to the point is represented as s. The k is the constant mentioned in Section 3.2.1.

$$OKS = e^{(-\frac{d^2}{2s^2k^2})} \tag{3.7}$$

Then, the predicted points are assigned to the nearest ground truth points. If the OKS of the pair of points is lower than a threshold, the pair is not considered as near. The predicted points, matched with the ground truth, are added to the True Positive (TP) in the next step. Next, the non-assigned predicted points are categorized as the False Positive (FP). The FP points are also called ghost-detected points. The ground truth points that do not match any predictions are counted as the False Negative (FN). Then, the number of each category is accumulated through frames. In the end, the average precision (AP) is computed using Equation (3.8).

$$AP = \frac{\Sigma_t TP_t}{\Sigma_t TP_t + \Sigma_t FP_t} \tag{3.8}$$

Finally, the mean of each AP are calculated as the mAP. Figure 3.19 shows how to categorize points to each category in one body part.

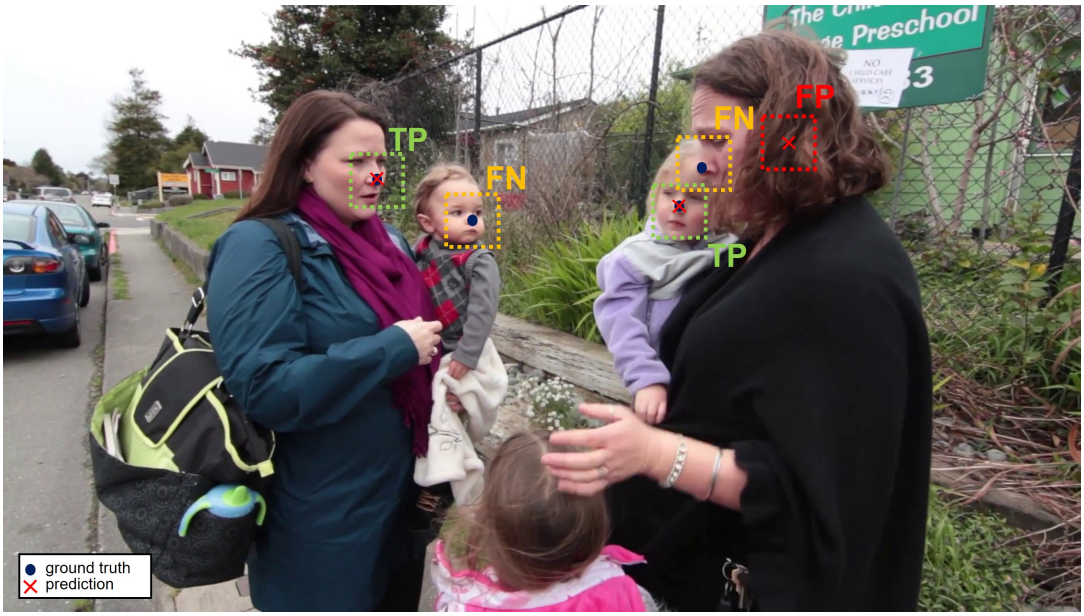


Figure 3.19: The True Positive (TP) is the predicted point that matches with the ground truth point. The ground truth point without the predicted point is called the False Negative (FN). The ghost point or the False Positive (FP) is the predicted point, which is not annotated in the ground truth.

3.5.3 Multiple Object Tracking Accuracy (MOTA)

Similar to the mAP, the Multiple Object Tracking Accuracy [61] (MOTA) is also calculated separately in each body part. Thus, the MOTA tells how well the model detects and tracks human body parts. In addition to the calculation for the mAP, the identification number is assigned to each point. The calculation for the MOTA is done by Equation (3.9).

$$MOTA = 1 - \frac{\Sigma_t(FN_t + FP_t + IDS_t)}{\Sigma_t GT_t} \tag{3.9}$$

In each frame, the matching of predicted points and ground truth points is done as same as the calculation for mAP. On top of counting TP, FP, and FN in each frame t, each matched predicted point's identification number is compared to the previously assigned number of the ground truth point. If the numbers are not the same, the Identification Switch (IDS) is added. The GT_t is the number of ground truth points in frame t. As shown in Equation (3.9), the MOTA can be a negative number.

4
Results

4.1 Multi-frame person pose estimation

In this task, both single-person videos and multi-person videos are available for benchmarking. Table 4.1 shows the mAP and the frames per second (FPS) of the single-person videos separated models. The scores in the table are the average values. The mAP for each joint is not shown in this thesis. Table 3.4 can interpret the identification number of each joint (Joint ID) to the human joint. The Flow Track is the fastest model in this multi-frame single-person pose estimation task with 1.77 frames per second and 78.9% mAP. The slowest and the least accurate model is the Joint Flow. The Joint Flow reaches only 41.2% mAP on speed 0.29 frames per second. The Light Track, designed to be the lightest model, is slower than the Flow Track and the Face-to-pose Track. However, the mAP of the Light Track is the highest score at 81.3%. The reason and observation are discussed in Section 5. The Face-to-pose Track, developed in this thesis, is in second place in speed and the mean average precision compared to the state-of-the-art models on the edge device—the model estimates human pose with 79.3% mAP at 1.56 frames per second. Figure 4.1 and Figure 4.2 show the detection results of four models.

Model	FPS	mAP
Flow Track	**1.77**	78.9%
Light Track	1.52	**81.3%**
Joint Flow	0.29	41.2%
Face-to-pose Track	1.56	79.3%

Table 4.1: Precision and speed of each model in terms of Frames per second and mean Average Precision on Single-person videos in Multi-frame person pose estimation task. The shown scores are the average value of every human body parts.

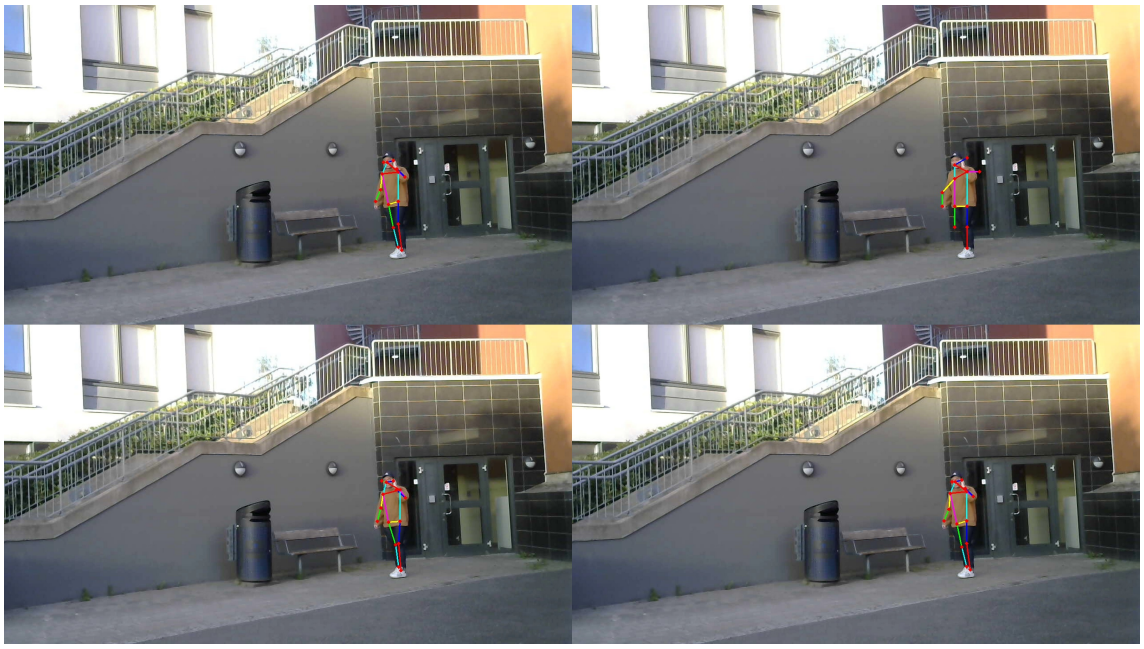


Figure 4.1: The result of multi-frame person pose estimation from Flow Track, Joint Flow, Light Track, and Face-to-pose Track is shown in this figure from top left, top right, bottom left, and bottom right. The input video is the single-human video.

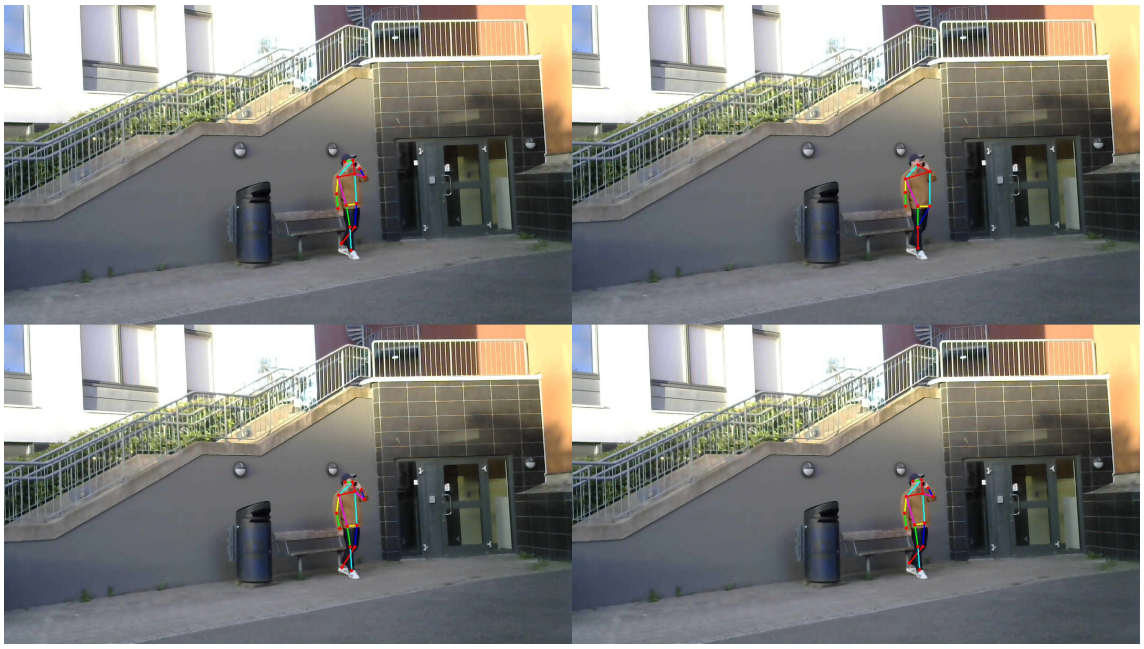


Figure 4.2: Another result of Flow Track, Joint Flow, Light Track, and Face-to-pose Track in multi-frame person pose estimation tasks, performed on single-human video, is shown on the figure's top left, top right, bottom left, and bottom right.

There are three to five persons in each frame for the multi-person videos, which increases the difficulty of the estimation and the processing time. Table 4.2 shows the mAP and the speed of the multi-person videos in the same pattern as the single-person videos. Flow Track's speed dropped to 0.74 frames per second with 74.5% mAP. At the same precision, the Face-to-pose Track runs at 0.54 frames per second. Thus, the Flow Track and the Face-to-pose Track share the highest mAP in this task. The Light Track with 70% mAP is faster than the Joint Flow with 27% mAP. The Light Track's speed and the Joint Flow's speed are 0.46 frames per second and 0.29 frames per second. The examples of predictions from four models are illustrated in Figure 4.3 and Figure 4.4.

Model	FPS	mAP
Flow Track	**0.73**	**74.5%**
Light Track	0.46	70.0%
Joint Flow	0.29	27.0%
Face-to-pose Track	0.535	**74.5%**

Table 4.2: Speed and precision of each model in terms of Frames per second and mean Average Precision on Multi-person videos in Multi-frame person pose estimation task. The shown scores are the average value of every human body parts.

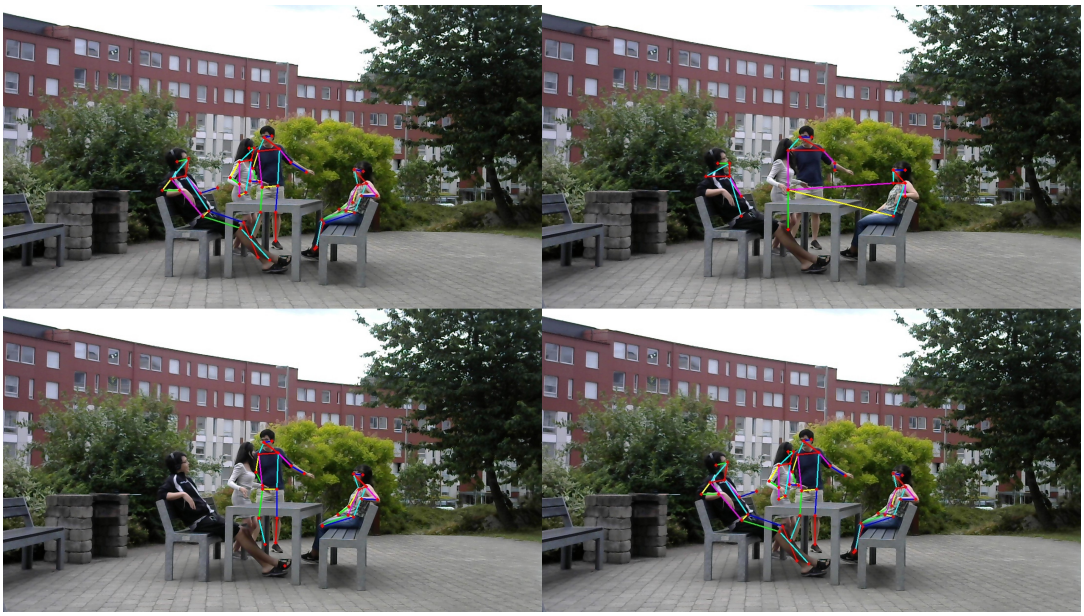


Figure 4.3: The result of multi-frame person pose estimation from Flow Track, Joint Flow, Light Track, and Face-to-pose Track is shown in this figure from top left, top right, bottom left, and bottom right. The input video is the multi-human video.

Figure 4.4: Another result of Flow Track, Joint Flow, Light Track, and Face-to-pose Track in multi-frame person pose estimation tasks, performed on multi-human video, is shown on the figure's top left, top right, bottom left, and bottom right.

4.2 Multi-person pose tracking

The Multi-person pose tracking task is to show how accurate each model is in assigning the identification numbers to detected human body parts. The scores of each model are shown in terms of Frames per second (FPS) and Multiple Object Tracking Accuracy (MOTA). The calculation is explained in Section 3.5. Similar to the Multi-frame person pose estimation task, every score is the mean value of each human body part. The scores are shown in Table 4.3. The Flow Track is the fastest and the most accurate model in this task, with 42.5% MOTA at 0.73 frames per second. The Light Track's accuracy, which is 41.0% MOTA, is not far from the Flow Track's accuracy. However, the Light Track is almost 40% slower than the Flow Track. With 35.5% MOTA at 0.535 frames per second, the Face-to-pose Track is slower than Light Track, but the Face-to-pose Track is less accurate than the Light Track. The Joint Flow performs with 21.5% MOTA at 0.29 frames per second. Figure 4.5 and Figure 4.6 are examples of results from each model.

Model	FPS	MOTA
Flow Track	**0.73**	**42.5%**
Light Track	0.46	41.0%
Joint Flow	0.29	21.5%
Face-to-pose Track	0.54	35.5%

Table 4.3: Accuracy and speed of each model in terms of Multiple Object Tracking Accuracy and Frames per second on Multi-person videos in Multi-person pose tracking task. The shown scores are the average value of every human body parts.

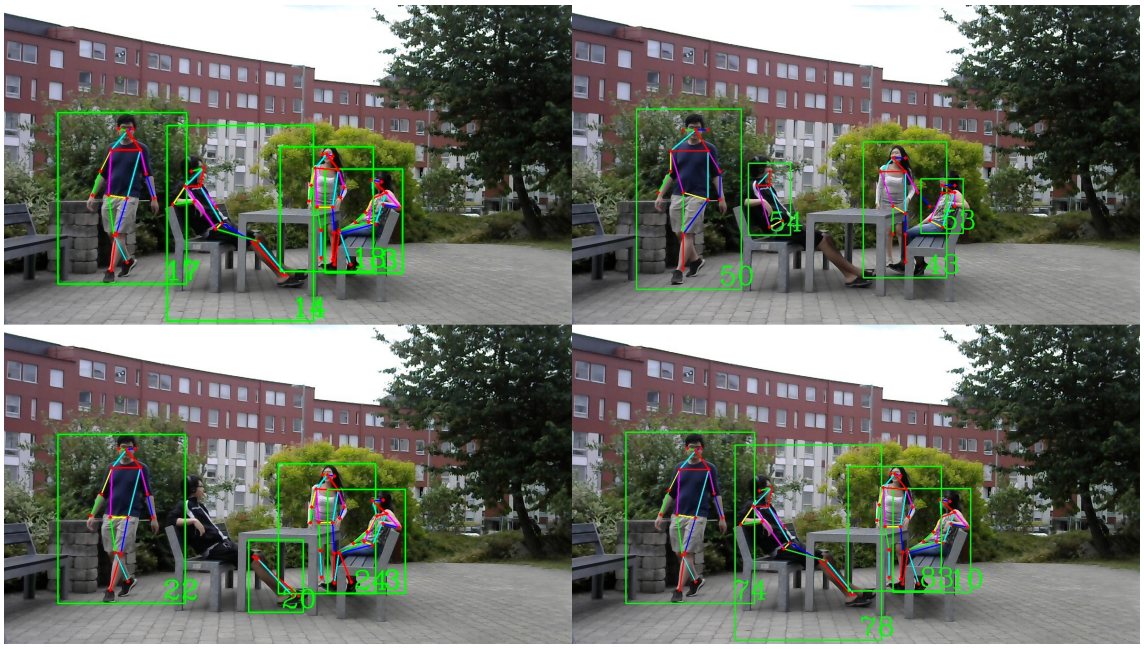


Figure 4.5: The result of multi-person pose tracking from Flow Track, Joint Flow, Light Track, and Face-to-pose Track is shown in this figure from top left, top right, bottom left, and bottom right.

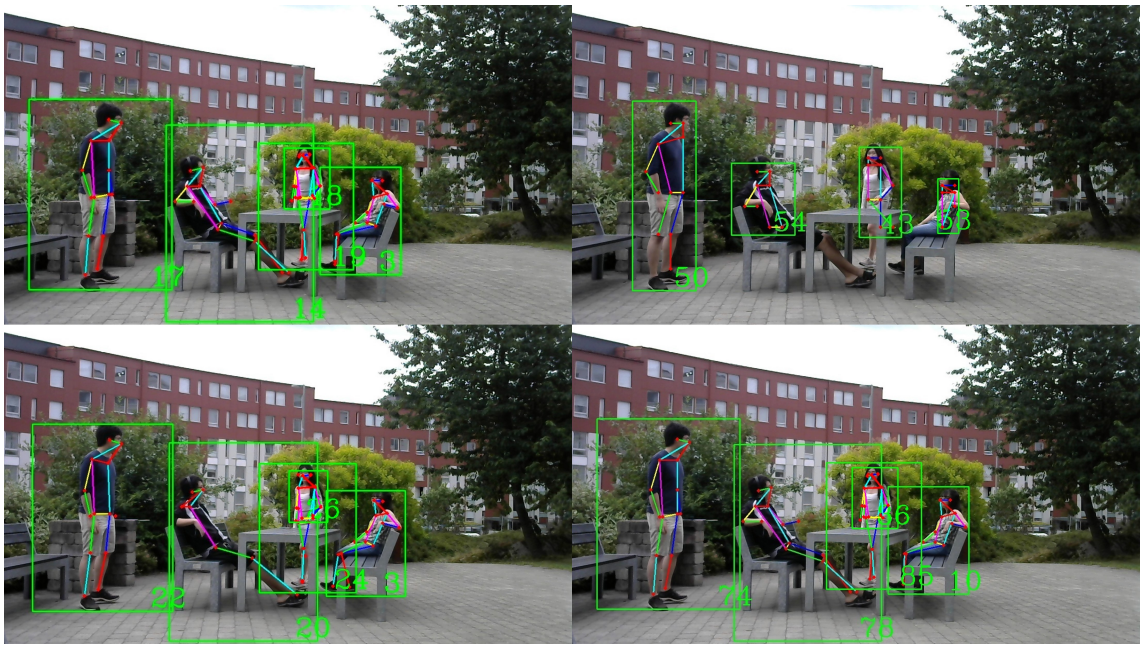


Figure 4.6: Another result of Flow Track, Joint Flow, Light Track, and Face-to-pose Track in multi-person pose tracking tasks is shown on the figure's top left, top right, bottom left, and bottom right.

4.3 Real-time pose tracking

In this thesis, the process that image acquisition and pose tracking are performed simultaneously is called Live mode. In Live mode, the FPS represents the processing speed for each model, as shown in table 4.4. Each model is evaluated by running Live mode with the HD USB camera, and the average FPS is calculated when each model reaches 150 frames. The mAP and the MOTA show how successful each model is in estimating human joints' location and tracking humans.

Model	FPS	mAP	MOTA
Flow Track	**1.97**	89.9%	80.1%
Light Track	1.82	**92.1%**	**85.5%**
Joint Flow	0.25	70.6%	56.2%
Face-to-pose Track	1.45	90.5%	77.7%

Table 4.4: Speed and Success rate of each model in terms of Frames per second, mAP, and MOTA in Live mode.

Flow Track is the fastest model in this thesis, with 1.97 FPS. The model track human acting in different poses with high performance. Since around 2 FPS allow a human to move significantly, the Flow Track assigns a new identification number to the same human if the human move very fast and the pose changes from the previous pose beyond the threshold. A critical feature of the Flow Track is that the model memorizes the pose from the first frame and can track humans even humans leaves the frame in some duration. Figure 4.7 shows the tracking result of Flow Track in sequence.

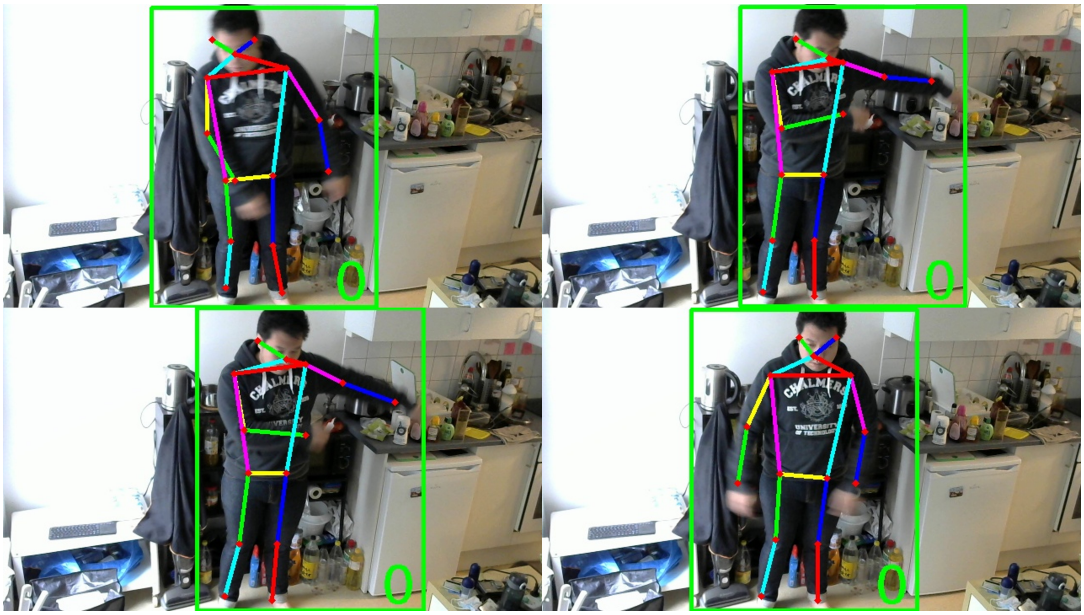


Figure 4.7: The sequence of Flow Track's result is shown from the figure's top left, top right, bottom left, and bottom right respectively.

Similar to Flow Track, the Light Track performs at 1.82 FPS. The performance is also similar. In the case of significant pose changes, the Light Track shows better tracking efficiency than the Flow Track because the Light Track uses the bounding box for tracking. When humans change pose quickly but do not change the position, as shown in Figure 4.8, the Light Track accurately tracks humans. In contrast, the Light Track is not performing well if humans change position with the high velocity.

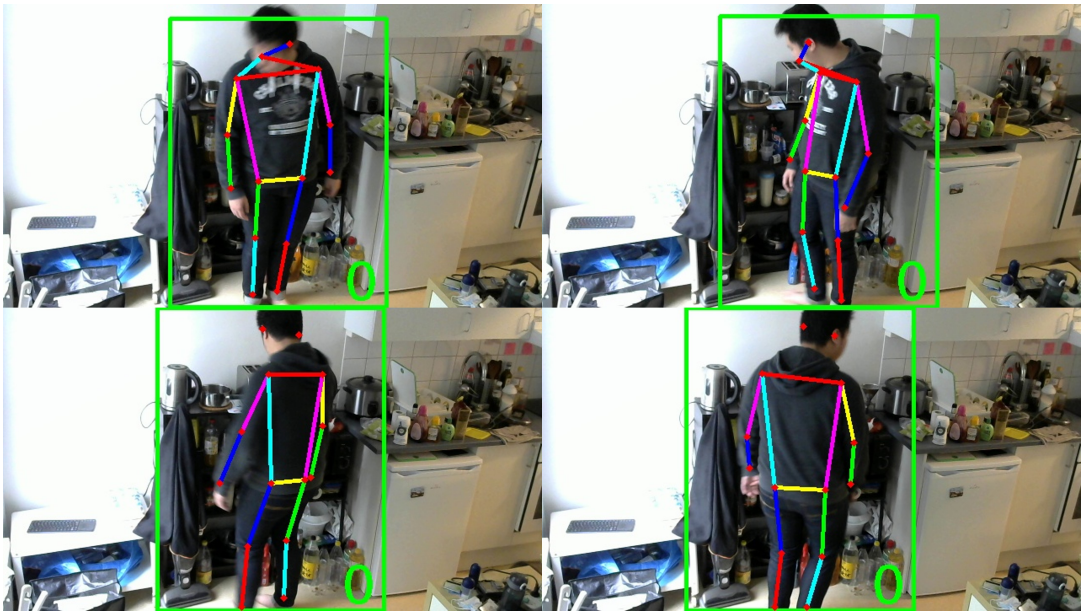


Figure 4.8: The sequence of Live Track's result is shown from the figure's top left, top right, bottom left, and bottom right respectively.

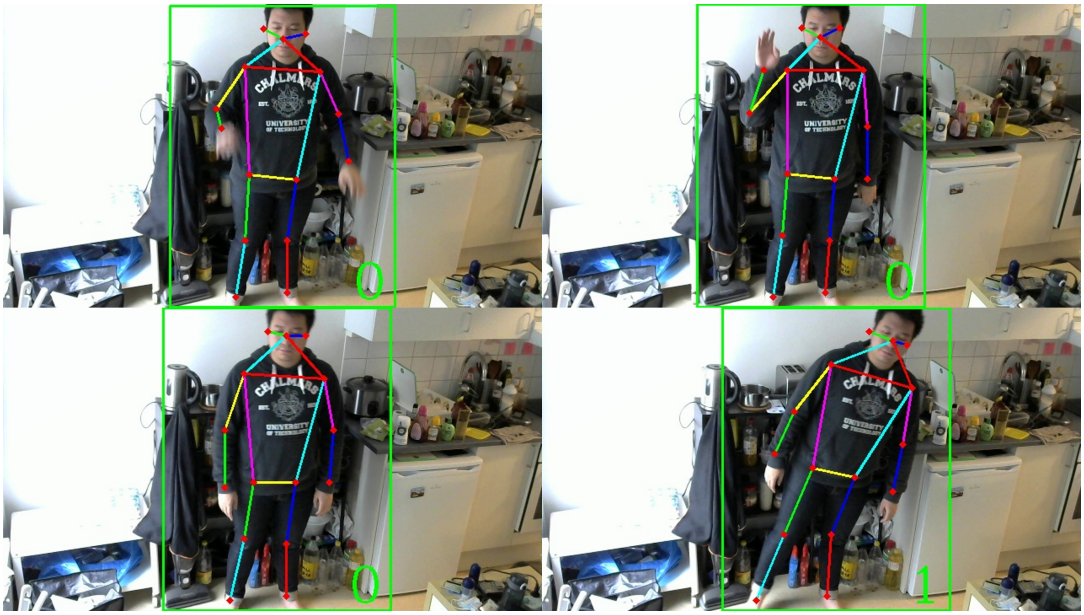


Figure 4.9: The sequence of Face-to-pose Track's result is shown from the figure's top left, top right, bottom left, and bottom right respectively.

The limitation of the Face-to-pose Track is the visibility of the face. If the face is not detected, the human is defined as a new person. Similar to the significant pose changes, if the face looks different from the previous frame, the identification number changes. Besides those two limitations, the Own Tack shows an outstanding performance. The model handles the significant pose changing case and the location changing case appropriately. Figure 4.9 shows the performance of the Face-to-pose Track in the pose changing case.
The Joint Flow is the model that does not perform well in this mode. Figure 4.10 shows four frames that are captured next to each other. As can be seen, the poses change significantly due to the speed of the Joint Flow. The model performs Live mode at 0.25 FPS which is the slowest in every model. The Joint Flow crashes after the Live mode reaches around 50 frames. Therefore, the model's speed is calculated until the crashed frame.

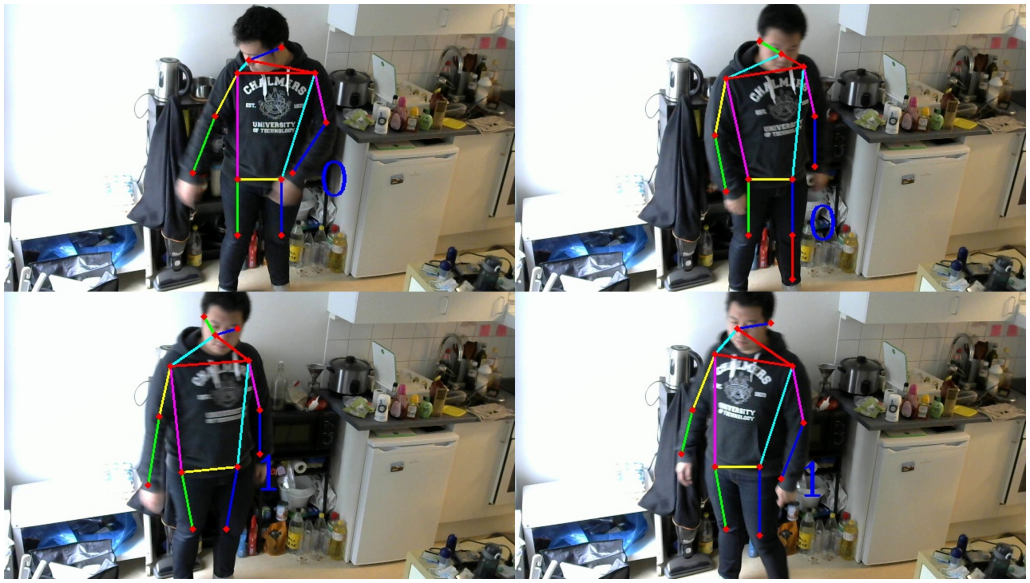


Figure 4.10: The sequence of Joint Flow's result is shown from the figure's top left, top right, bottom left, and bottom right respectively.

5

Discussion

First, the evaluation from pre-recorded videos is discussed. The performances of the state-of-the-art (SOTA) models evaluated in this thesis are different from the leaderboard of the Posetrack challenge [43] based on two main reasons. The first reason is that the scores on the leaderboard are evaluated on the high-performance computer with different frameworks and settings. Since this thesis evaluates the models in the edge computing environment, both frameworks and settings are changed to the available resources on edge devices.

The pre-trained models from original works are not available for this thesis due to the dependencies. Most of the models are developed in TensorFlow 1 framework [42], which is the discontinued framework from Google. In this thesis, the models are built and trained on TensorFlow 2. Therefore, the training time for each model is different from the original work. It is worth mentioning that this thesis spends approximately the same amount of training time on each new-built model to control the training time factor. From Table 4.1, Table 4.2, and Table 4.3, the speed of SOTA models drops significantly regarding the computation power of the Nvidia Jetson Nano. Besides the speed, mAPs are similar to the scores in the leaderboard, and MOTAs are slightly lower than the scores.

As shown in tables in Section 4, the Flow Track is the fastest model in every task. The mAP and MOTA of the Flow Track are also at the top of the tables. With the simple architecture of the model, the Flow Track is an excellent choice to be a baseline for further improvement as this thesis has done on the Face-to-pose Track. The disadvantage of the Flow Track happens when there are two humans act the same pose. The skeletons of both humans look the same, and the model cannot separate them.

The light pose tracking model or Light Track has several issues to be discussed. Firstly, the Light Track is slower than the Flow Track and the Face-to-pose Track. From the Light Track's concept, the model relies on the matrix calculation instead of the neural networks because designers believe that the neural networks require the long processing time. The hypothesis is proved in [51] on the high-quality computer. However, the neural networks can be implemented with equal or higher speed comparing to the matrix calculation on the Nvidia Jetson Nano with the TensorRT framework. Next, the Light Track's inputs are bigger than the inputs of the Flow Track and the Face-to-pose Track. The larger image contains a lot more information or features. Thus, the human pose estimator module of the Light Track is more efficient than the others.

The Light Track's mAP is lower than the Flow Track and the Face-to-pose Track in the multi-person pose estimation, even though the mAP of the Light Track is the best in the single-person pose estimation. The reason is that the mAP is related to the human detection part as well. If the human is not detected, the pose estimator module cannot predict any joints of that human. The human detection module of the Light Track uses the bounding box from the previous frame, and there is a high chance that the person moves out from the scaled bounding box or another person comes into the box, which causes errors. According to the bounding boxes, the Light Track tracks humans from the IoU of each bounding box. If two humans pass by each other, the identification number might switch or be lost in the frame.

The Joint Flow, which is the only bottom-up approach in this thesis, is not showing its best performance. The model is in the last place in speed, mAP, and MOTA. The observation for this problem is that the bottom-up require more data and more training time to reach the state-of-the-art level. Since the COCO dataset does not contain the relation of 2 frames, the PoseTrack dataset, which contains video sequences, is the only training dataset for the model. In the model's investigation, the predictions for PAF and TFF are not very accurate compared to the ground truths. Another observation is that the Joint Flow requires a lot of matrix calculation, linear algebra calculation, and loop structure to assemble humans and compute the tracking vectors. As discussed in the Light Track, the operation of the neural network performs with the impressive speed on the Jetson Nano. With the high-performance computer, the Joint Flow can be one of the excellent choices for human pose tracking, but it is not matched with the Jetson Nano.

The Face-to-pose Track, developed for this thesis, aims to use the benefit of parallel computation in TensorRT since the size and the processing time of neural networks are reduced with the framework. As can be seen in the results, the Face-to-pose Track reaches the first or second rank in both speed and mAP. Furthermore, the model is third in the multi-person pose tracking task, but the score is not far from the SOTA models. The similarity prediction from human faces works fast and efficiently, but there are still some problems. For example, the situation that faces are not detected, the siamese model can not tell the difference between two persons.

The idea of changing from face similarity to image similarity can be added to future work. Instead of feeding cropped face images to the model, the image of the detected human can be fed to the model to predict the similarity. The hypothesis is that there are more features to be used and to avoid the non-detected case. More memory can be added as well. The model can save results from many frames instead of only the previous frame. This future work can be helpful to avoid the case that humans change their head orientation very fast. For example, the head changes from the look towards the camera to the ground in one frame, which the siamese model considers two different persons.

The similarity is not fixed to only one measurement. In future work, the study of the combination of measurements must be added. For example, the average OKS calculation can be used along with the face similarity to compensate for the face missing or same skeleton situations. The speed is the trade-off for this method, but it can be more accurate in the applications that speed is not the critical factor. In

commercial applications, the live mode tends to be implemented in various kinds of situations. The speed of the models is doubted in live mode. Some applications that high speed does not require, such as the workers counting in the factory. In this case, the workers stay at their stations and do not move much. The speed at one or two frames per second can handle the application. If the speed is needed, the high-performance edge devices can be selected. For example, more choices have higher computation power than the Jetson Nano from Nvidia, such as Jetson TX2 [62] or Jetson Xavier NX [63].

6

Conclusion

In this thesis, top-down pose tracking approaches and bottom-up pose tracking approaches were studied to answer research questions. Public datasets were used to train neural network models and our new dataset was used to benchmark approaches. Our new dataset was collected with Nvidia Jetson Nano and HD USB camera to simulate the edge computing environment. Flow Track and Light Track are the state-of-the-art top-down pose tracking approaches to the Posetrack challenge, while Joint Flow is the state-of-the-art bottom-up pose tracking approach. The approach, which was developed for this thesis aiming to suit the edge computing applications, is called Face-to-pose Track. The Face-to-pose track aims to minimize the usage of matrix calculation and maximize the usage of neural network computing with the TensorRT framework.Each approach was developed on TensorFlow 2 framework and trained with the public datasets, which are the COCO dataset and the PoseTrack dataset. The size of each model was too large to deploy on Nvidia Jetson Nano. Thus, they were converted to the TensorRT framework. Approaches were benchmarked in three tasks: Multi-frame person pose estimation, Multi-person pose tracking, and Live tracking. The Mean Average Precision (mAP) and the Multiple Object Tracking Accuracy (MOTA) showed the approach's performance in predicting human joints' locations and tracking human body parts respectively.

The performance of the top-down pose tracking approaches compared to the bottom-up pose tracking approaches in human pose estimation and human pose tracking based on the Mean Average Precision and Multiple Object Tracking Accuracy is the first research question. Three top-down pose tracking approaches outperformed bottom-up pose tracking approaches both in Multi-frame person pose estimation and Multi-person pose tracking. The input image's size and the training data were keys in this part. The edge device's memory was smaller than the desktop computer. Therefore, the size of the neural network model, including the input image's size, was reduced from the original model. While the top-down approaches detect a human from the human detector module, the bottom-up approaches predict human joints' location, predict joint relationship, and assembly into a human. The Part Affinity Field and the Temporal Flow Field, which were used to state joint relationships in the frame and between frames, were not accurate because of the quality of the input image and the small number of training data.

The speed, in terms of frames-per-second (FPS), of top-down pose tracking approaches and bottom-up pose tracking approaches on edge devices was asked in the second research question. All top-down pose tracking approaches studied in this thesis were faster than the bottom-up pose tracking approach significantly. The

TensorRT framework, which was developed to optimize the parallel programming model on the GPU, reduced neural network computing time. While both top-down approaches and bottom-up approaches required a similar amount of neural network computing, the bottom-up approaches needed more matrix calculation to assemble and track humans compared to the top-down approaches. A matrix calculation is not the strong advantage of edge devices.

The approach to optimize and improve the accuracy and speed of pose tracking on edge devices is the next research question. As mentioned in the previous paragraph, the TensorRT framework optimized neural network computing. The Flow Track and the Light Track tracked humans by the Average Object Keypoint Similarity and the Intersection Over Union of bounding boxes. Both similarities are not using neural network computing. Thus, the Face-to-pose Track was developed to maximize the usage of neural networks. The results showed that the Face-to-pose Track performed excellently together with the state-of-the-art Flow Track both in accuracy and speed.

The effect of speed, in terms of frames-per-second, on the accuracy, based on the visualization from the edge device and USB camera in real-time, where image acquisition and pose tracking are performed simultaneously is the last research question. As can be seen in the Live mode's result, the processing speed is the big problem of the Live mode. The slow speed has a high potential to decrease the performance of each model. The bottom-up pose tracking approach suffered from the speed significantly. With approximately 4 seconds per frame, the differences in human poses between the 2 frames were too large to track.

In future work, the combination of various similarity scorings can be the key to optimize the performance of the human pose tracking approaches both in speed and accuracy. As an example, the face similarity in the Face-to-pose Track lost human tracking when the faces were not visible and the Average Object Keypoint Similarity predicted the wrong human identification when two humans were in the same pose. One similarity score can be used to compensate for others' weaknesses. An upgrade in hardware is also possible. There are several kinds of edge devices in the market. A suitable device can be chosen for the specific application.

Bibliography

[1] Michael I. Jordan, "Serial order: A parallel distributed processing approach," *Advances in Psychology*, vol. 121, 1997.

[2] Sepp Hochreiter, Jürgen Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, 1997.

[3] S Indolia, AK Goswami, SP Mishra, P Asopa, "Conceptual understanding of convolutional neural network-a deep learning approach," *Procedia Computer Science*, vol. 132, 2018.

[4] B. Patil, "Deep learning for natural language processing," *International Journal for Research in Applied Science and Engineering Technology*, vol. 9, pp. 854–857, 04 2021.

[5] M. Kozlenko, I. Lazarovych, and M. Kuz, "Deep learning approach to signal processing in infocommunications," 09 2020.

[6] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, pp. 1–21, 01 2019.

[7] V. Rajinikanth, E. Priya, H. Lin, and F. Lin, *Deep Learning for Medical Image Processing*, 12 2020, pp. 147–183.

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278 – 2324, 12 1998.

[9] M. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *European Conference on Computer Vision(ECCV)*, vol. 8689, pp. 818–833, 01 2013.

[10] D. Hubel and T. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, pp. 215–43, 04 1968.

[11] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Convolutional neural networks for large-scale remote sensing image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, 09 2016.

[12] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[13] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning." vol. 930, 06 1995, pp. 195–201.

[14] Y. Zhou, H. Wang, F. Xu, and Y.-Q. Jin, "Polarimetric sar image classification using deep convolutional neural networks," *IEEE Geoscience and Remote Sensing Letters*, vol. PP, pp. 1–5, 11 2016.

[15] C. Lemaréchal, "Cauchy and the gradient method," *Documenta Mathematica*, 01 2012.

[16] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[17] A. Graves, "Generating sequences with recurrent neural networks," 08 2013.

[18] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. Zitnick, "Microsoft coco: Common objects in context," vol. 8693, 04 2014.

[19] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, and V. Ferrari, "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale," *International Journal of Computer Vision*, vol. 128, 03 2020.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li, "Imagenet: a large-scale hierarchical image database," 06 2009, pp. 248–255.

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 06 2015.

[22] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, pp. 1–1, 07 2018.

[23] I. Khokhlov, E. Davydenko, I. Osokin, I. Ryakin, A. Babaev, V. Litvinenko, and R. Gorbachev, "Tiny-yolo object detection supplemented with geometrical data," 05 2020, pp. 1–5.

[24] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," 07 2017, pp. 6517–6525.

[25] A. Bochkovskiy, C.-Y. Wang, and H.-y. Liao, "Yolov4: Optimal speed and accuracy of object detection," 04 2020.

[26] G. Jocher, A. Stoken, J. Borovec, NanoCode012, A. Chaurasia, TaoXie, L. Changyu, A. V, Laughing, tkianai, yxNONG, A. Hogan, lorenzomammana, AlexWang1900, J. Hajek, L. Diaconu, Marc, Y. Kwon, oleg, wanghaoyang0106, Y. Defretin, A. Lohia, ml5ah, B. Milanko, B. Fineran, D. Khromov, D. Yiwei, Doug, Durgesh, and F. Ingham. (2021, Apr.) ultralytics/yolov5: v5.0 - YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integrations. [Online]. Available: https://doi.org/10.5281/zenodo.4679653

[27] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, 06 2015.

[28] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask r-cnn," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, pp. 1–1, 06 2018.

[29] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 11 2013.

[30] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, "2d human pose estimation: New benchmark and state of the art analysis," 06 2014.

[31] J. Wang, S. Tan, X. Zhen, S. Xu, F. Zheng, Z. He, and L. Shao, "Deep 3d human pose estimation: A review," *Computer Vision and Image Understanding*, vol. 210, p. 103225, 05 2021.

[32] A. Toshev and C. Szegedy, "Deeppose: Human pose estimation via deep neural networks," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 12 2013.

[33] R. Guler, N. Neverova, and I. Kokkinos, "Densepose: Dense human pose estimation in the wild," 06 2018, pp. 7297–7306.

[34] H.-S. Fang, S. Xie, Y.-W. Tai, and C. Lu, "Rmpe: Regional multi-person pose estimation," 10 2017, pp. 2353–2362.

[35] Z. Cao, G. Martinez, T. Simon, S.-E. Wei, and Y. Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, pp. 1–1, 07 2019.

[36] L. Pishchulin, E. Insafutdinov, S. Tang, B. Andres, M. Andriluka, P. Gehler, and B. Schiele, "Deepcut: Joint subset partition and labeling for multi person pose estimation," 06 2016, pp. 4929–4937.

[37] Y. Zhong, J. Wang, J. Peng, and L. Zhang, "Anchor box optimization for object detection," 03 2020, pp. 1275–1283.

[38] S. Ren, K. He, R. Girshick, X. Zhang, and J. Sun, "Object detection networks on convolutional feature maps," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, 04 2015.

[39] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," 07 2017, pp. 936–944.

[40] P. P. Ray, "An introduction to dew computing: Definition, concept and implications," *IEEE Access*, vol. 6, pp. 723–737, 2018.

[41] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. Nikolopoulos, "Challenges and opportunities in edge computing," 11 2016.

[42] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[43] M. Andriluka, U. Iqbal, E. Insafutdinov, L. Pishchulin, A. Milan, J. Gall, and B. Schiele, "Posetrack: A benchmark for human pose estimation and tracking," 06 2018, pp. 5167–5176.

[44] Jetson nano developer kit. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[45] C270 hd webcam. [Online]. Available: https://www.logitech.com/en-us/products/webcams/c270-hd-webcam.960-000694.html?crid=34

[46] J. Tompson, R. Goroshin, A. Jain, Y. Lecun, and C. Bregler, "Efficient object localization using convolutional networks," 06 2015, pp. 648–656.

[47] B. Xiao, H. Wu, and Y. Wei, "Simple baselines for human pose estimation and tracking," in *European Conference on Computer Vision (ECCV)*, 2018.
[48] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 06 2016, pp. 770–778.
[49] W. Shi, J. Caballero, L. Theis, F. Huszar, A. Aitken, C. Ledig, and Z. Wang, "Is the deconvolution layer the same as a convolutional layer?" 09 2016.
[50] R. Girdhar, G. Gkioxari, L. Torresani, M. Paluri, and D. Tran, "Detect-and-track: Efficient pose estimation in videos," 06 2018, pp. 350–359.
[51] G. Ning, J. Pei, and H. Huang, "Lighttrack: A generic framework for online top-down human pose tracking," 06 2020, pp. 4456–4465.
[52] A. Doering, U. Iqbal, and J. Gall, "Joint flow: Temporal flow fields for multi person tracking," *BMVC 2018*, 2018.
[53] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv 1409.1556*, 09 2014.
[54] Nvidia tensorrt. [Online]. Available: https://developer.nvidia.com/tensorrt
[55] Z. Bukovcikova, D. Sopiak, M. Oravec, and J. Pavlovicova, "Face verification using convolutional neural networks with siamese architecture," 09 2017, pp. 205–208.
[56] tensorrt demos. [Online]. Available: https://github.com/jkjung-avt/tensorrt_demos
[57] S. Ruder, "An overview of gradient descent optimization algorithms," 09 2016.
[58] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
[59] Nvidia tensorrt documentation. [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html
[60] L. Pishchulin, E. Insafutdinov, S. Tang, B. Andres, M. Andriluka, P. Gehler, and B. Schiele, "DeepCut: Joint Subset Partition and Labeling for Multi Person Pose Estimation," *CVPR 2016*, 2016.
[61] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: the clear mot metrics," *EURASIP J. Image Vide.08*, 2008.
[62] Jetson tx2 module. [Online]. Available: https://developer.nvidia.com/embedded/jetson-tx2
[63] Jetson xavier nx. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx

DEPARTMENT OF Eletrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY