Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Haakon Tideman Kanter
Omar Tveiten Sheikh
Aleksander Tosbakken

Hand gesture recognition and interactivity in Pexip video conference calls

Bachelor's thesis in Computer Science
Supervisor: Ole Christian Eidheim
May 2023

NTNU
Norwegian University of
Science and Technology

Haakon Tideman Kanter
Omar Tveiten Sheikh
Aleksander Tosbakken

Hand gesture recognition and interactivity in Pexip video conference calls

Bachelor's thesis in Computer Science
Supervisor: Ole Christian Eidheim
May 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

This thesis covers the development of a hand gesture detector, developed for Pexip. There are two main parts of this process, namely an object detection model, as well as a GStreamer element with a detection processing algorithm. The hand gesture detection model was made with prominently recognized object detection architectures. Among the architectures explored were YOLO, SSD and Faster R-CNN. All of the object detection models were trained on the HaGRID dataset, which includes 18 different gestures.

Some of the trained models achieved a mAP@50 score of over 0.9, while also maintaining a sufficient framerate. The final selection of models consisted only of different versions of YOLO models. These models had framerates ranging from 3.3 FPS to 26.3 FPS.

The second part of this thesis was the development of a GStreamer element with a detection processing algorithm. As Pexip requires high performance, this was done in C++. This element was responsible for emitting a "raised hand" signal, which would trigger a "raised hand" functionality within the Pexip webapp, raising a banner and notifying other participants. It was also important that this element had high performance, as responsiveness and minimal resource use in terms of CPU and memory was desired. This element also contained debugging features, allowing quick and easy experimentation.

The final result was a gesture detector, which is currently in one of Pexip's GitHub branches. The detector notifies other users within a video call that a participant has raised their hand. As the model contains a multitude of hand gestures, the gesture detector is easily extensible to help provide additional desired features. The detector serves as a strong foundation for not only using the "raise hand" feature, but can also be extended for a multitude of other use cases.

Sammendrag

Denne oppgaven omfatter utviklingen av en håndgests-detektor, som ble utviklet for Pexip. Det er to hoveddeler til denne prosessen, en objektdetekteringsmodell og et GStreamer element med en prosesseringsalgoritme for deteksjoner. Håndgest-modellen ble laget med bredt anerkjente objekt-detektsjons arkitekturer. Arkitekturene som ble utforsket var YOLO, SSD og Faster R-CNN. Alle objektdeteksjonsmodellene ble trent på HaGRID datasettet, som inneholder 18 forskjellige håndgester.

Resultatet av å trene disse modellene var flere modeller med mAP@50 resultater over 0.9, som også hadde akseptabel eller god ytelse. De fullstendig implementerte modellene hadde rammerater fra 3.3 FPS til 26.3 FPS.

Den andre delen av denne oppgaven var utviklingen av et GStreamer element med en prosesseringsalgoritme for deteksjoner. Dette var hovedsakelig implementert i C og C++, ettersom god ytelse var et krav. Dette elementet var ansvarlig for å sende ut et "oppreist hånd" signal, som igjen kunne bli vist i Pexips webapplikasjon. Det var viktig at dette elementet hadde god ytelse, ettersom responsivitet og minimal ressursbruk var foretrukket. I tillegg til dette ble elementet utviklet med en håndfull av debuggingsfunksjoner, slik at å eksperimentere, debugge og endre på parametere var så enkelt som mulig.

Sluttresultatet var en "håndgest" detektor, som for tiden er i en branch i Pexips GitHub repository. Detektoren varsler andre deltakende i en videokonferanse om at en deltaker har løftet hånden sin. Siden modellen har flere håndgester, er den lett å utvide dersom flere funksjoner skulle ønskes.

Preface

This bachelor's thesis is written for Pexip, a company that provides video conference services, as a part of our bachelor's degree in Computer Science at Norwegian University of Science and Technology (NTNU). It has been a valuable experience which has provided the team members with insight and experience within several domains of a production environment. It has given valuable insight in how to navigate a professional work environment, and a more detailed understanding of deep learning, object detection architectures, and the use of C++.

We would like to express our gratitude to Håvard Graff and Frode Olsen from Pexip for their invaluable assistance in constructing this thesis statement and providing excellent guidance throughout the entire process. Furthermore, we extend our heartfelt thanks to the rest of the dedicated Pexip team at Lysaker for their support and active engagement throughout the process.

We also wish to thank Ole Christian Eidheim for good guidance in this thesis, with valuable feedback and support. In addition to this, we want to express our gratitude for Idun access. We also want to thank family and friends for support along the way.

Haakon Tideman Kanter | Omar Tveiten Sheikh | Aleksander Tosbakken

Trondheim, May 21, 2023

Project description

Thesis question How can one leverage machine learning to create a seamless solution for various hand gestures in video conferences, allowing them to be used to interact with different features on relevant platforms?

Project description This project aims to use machine learning to create a solution that can recognize hand gestures in real-time, such that it can be used to interact with functionality available within the video-conference solution that Pexip provides. The solution will run as a background process during video conferences.

The goal is to develop a module that can be added to the Pexip video-conference platform. This module should be able to emit different signals based on detected hand gestures from participants in video conferences. The module should also be optimized enough to avoid causing performance issues for the servers that processes data from conferences.

Contents

Figures

Tables

Code Listings

Acronyms

ANN Artificial Neural Network. xv, 4–6

AP Average Precision. xv, 13–15

API Application Programming Interface. 23, 25, 30, 53, 54

bbox bounding box. 20, 21

CD continuous deployment. 22

CI continuous integration. 22

CNN Convolutional Neural Network. xv, 6, 7, 9, 15, 17, 18

COCO Common Objects in Context. 18, 21, 28, 29, 50

CPU Central Processing Unit. iii, xvii, 1, 37, 40, 45, 57, 59

CSV comma-separated values. 29

Faster R-CNN Faster Region-based Convolutional Neural Network. iii, v, xvi, 18, 27–30, 39, 43, 44, 47, 49, 54

FN false negative. 11, 12, 59

FP false positive. 11, 12, 33, 41, 48, 59

FPS frames per second. iii, v, xix, 40, 45, 47, 48

GPU Graphical Processing Unit. 10, 24, 29, 37, 40, 45, 52, 61

IoU Intersection over Union. xv, 12–17, 32, 43

JSON JavaScript Object Notation. 21

mAP Mean Average Precision. iii, v, xvii, 14, 15, 18, 27–30, 43, 47, 50, 54

NMS Non Maximum Suppression. 15, 18, 38

NN Neural Network. xxiii, 4, 5, 9, 17

NTNU Norwegian University of Science and Technology. vii, 24, 29

R-CNN Region Based Convolutional Neural Networks. 15

ReLU rectified linear unit. 9

ROI region of interest. 18

RPN Region Proposal Network. 18, 19

RWL result with lifespan. xv, 32–34

SSD Single Shot MultiBox Detector. iii, v, xv, xvi, 9, 15–17, 27–30, 39, 43, 44, 47, 54

TDD test-driven development. 24, 40

TF TensorFlow. 29, 30, 54

TFRecord TensorFlow Record. 29, 54

TN true negative. 11, 12

TP true positive. 11, 12

UI user interface. 1

VOC Visual Object Classes. 21, 29

XML eXtensible Markup Language. 21

YOLO You Only Look Once. iii, v, xvii, 15, 17, 18, 20, 27–29, 36, 38, 39, 41, 43–45, 47–50, 52, 57, 58, 69

Glossary

batch size The amount of samples that is propagated through a NN. xxiv, 29, 30, 38

bounding box A bounding box is a rectangle that surrounds an object, and specifies its position . xvi, 15, 17, 18, 20, 21, 27, 32, 33, 35, 37, 38, 49–51

confidence A value between 0 and 1 describing how certain a computer vision model is that its observation is correct, where 0 is the lowest possible value and 1 is the highest. 27

enum A data type in programming languages that represents a set of named values.. 31, 36

epoch A epoch is a single pass through the entire training dataset during the training phase of a model. . 29

feature extraction Feature extraction refers to extracting meaningful and informative features from raw input data. In computer vision, this often refers to extracting patterns from images. 18

flattened To be converted from a multi-dimensional array or tensor to a 1-dimensional array or tensor. xv, 6

inference The process of running data through a machine learning model and calculating an output. 18

label map Data structure or mapping that associates unique labels or identifiers with corresponding elements or categories. 30

model parameters Model parameters are the internal values that determine the behavior of the model. These are usually just numbers in a matrix. 18, 53, 60

model size The amount of parameters that a machine learning model consists of. 18

objectness The probability of an object existing within a given region of interest. 38

operator set A collection of defined operations. 24, 30, 37, 53

script A sequence of instructions or commands written in a programming or scripting language.. 29, 30, 50, 52, 53

step A step refers to a single update of the model's parameters during the training process, which occurs after a certain amount of data has passed through the model. This amount of data is referred to as the batch size. . 30, 52

transcoding Conversion of one type of digital media, such as video or audio, to another. 1

unflattened To be converted from a 1-dimensional array or tensor to a multi-dimensional array or tensor. 37

unstructured data All data that is not provided in a structured layout (such as databases). Some examples of unstructured data is video files, audio files, websites and articles. According to MongoDB, 80-90% of generated data and data used by organizations is unstructured data, and the amount of unstructured data is rapidly growing. [1] . 4

Chapter 1

Introduction

1.1 Background

In a Pexip video call, video and audio undergoes transcoding on the server-side to provide the best experience for each client. This gives Pexip a unique opportunity to analyze, interpret and optimize video and audio from each participant without utilizing the users limited resources, such as Central Processing Units (CPUs) and battery.

1.2 Motivation

In meetings, or in conferences and classrooms, a participant may raise their hand to get attention. In a Pexip video conference call, this is emulated by clicking a "raise hand" button. This will notify other participants that you want to speak. To further expand upon this concept, it is possible to utilize machine learning to detect and recognize hands and gestures. By detecting certain gestures, Pexip may be able to create a more seamless and interactive experience for their users by implementing a system that can trigger the same functionality as clicking the "raise hand" button when detecting a raised hand on live video. This can create a more immersive experience for a user who wants to speak, so they can naturally raise their hand to perform the "raise hand" functionality in a video conference. Additionally, not all devices which can be used in a Pexip call has the ability to display the user interface (UI) which allows a user to click the "raise hand" button. Detecting raised hands on video calls can directly help circumvent this obstacle.

Pexip is also capable of interoperating with other platforms, such as Zoom and Microsoft Teams. By detecting hands on live video, Pexip may be able to use their own "raise hand" functionality to trigger the same features on the other platforms, given that they acquire permission to do this.

1.3 Objectives

1.3.1 Primary

These are the main goals that were expected to be completed for the final solution.

- Integrate a machine learning model with Pexip's video conference platform.
- Create a basic gesture detection module capable of detecting specific gestures.
- Detect a raised hand with one video participant.
- Detect a raised hand with multiple video participants.

1.3.2 Secondary

In order to create the final solution, the following goals were not necessary, but were still important goals that could improve the overall solution.

- Find and train an object detection model that performs well.
- Integrate the model with good levels of optimization with respect to memory and CPU computation.

Chapter 2

Theory

This chapter gives an overview of various theoretical means which have been employed. It seeks to introduce, and help understand, the base for methodology in section 3.

2.1 Machine learning

Machine learning is a subset of artificial intelligence which is related to systems and algorithms that are able to learn from data and make predictions from said data [2]. These algorithms iteratively attempt to identify patterns within given data and learn from them. As the algorithm is exposed to more data, it is expected to perform better, given that there are patterns within the data. A popular and more formal definition for algorithms within the machine learning space was provided by Tom M. Mitchell:

> *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.* [3]

Supervised learning, unsupervised learning, and reinforcement learning are the primary types of machine learning approaches. Machine learning is useful for tasks that would be hard or borderline impossible to solve with traditional programming. Machine learning algorithms are capable of being highly complex, and have enormous amounts of applications, such as computer vision [4, 5], natural language processing [6], and robotics [7].

2.1.1 Classification

Classification is a subset of machine learning which aims to predict the correct label of given input data. Classification is a supervised machine learning method, which means that classification algorithms uses labeled data sets to train and measure performance of the model [8] . Data is often split into training and test sets, where the model will train on the training set and then be measured on its accuracy in the test set to ensure it is predicting accurately.

The application of classification is widespread. It is employed in tasks such as image recognition, text recognition, spam detection, and fraud detection, among others [8]. For instance, in image recognition, a classifier can be trained to detect the presence of an object in an image. Similarly, in natural language processing, a classifier can be used to classify the text into different topics or sentiments. The application of classification has enhanced the accuracy and efficiency of automated systems, making it an essential tool for many fields [9–11] .

2.1.2 Deep learning

Deep learning is a subset of machine learning which uses 2.1.3 Artificial Neural Networks (ANNs) with multiple layers to simulate the way the human brain behaves [2]. This type of learning is exceptionally valuable as it can interpret unstructured data, such as images, audio and unstructured text such as websites or articles [12]. Deep learning has drastically improved object recognition and object detection techniques, among many other fields [13]. Deep learning is used in a multitude of everyday challenges, such as providing digital assistants or preventing fraud. Deep learning algorithms continue to evolve, leading to noteworthy recent advancements in the field. Some examples of this is the emergence of tools capable of generating images from textual prompts (e.g., MidJourney [14], Dall-E [15], Stable diffusion [16]), and AI language models capable of generating human-like text, such as Chat Generative Pre-trained Transformer (ChatGPT) [17].

2.1.3 Artifical Neural Networks

An Artificial Neural Network (ANN), also occasionally shortened to just Neural Network (NN) is a fundamental component of modern machine learning algorithms. ANNs seek to replicate neural pathways in the human brain and are composed of three types of layers: input, hidden, and output [18]. A traditional ANN is visualized in Figure 2.1.

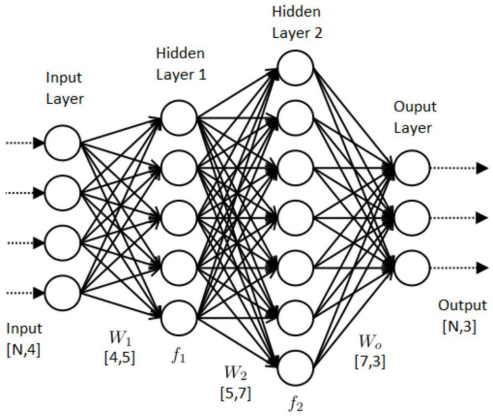


Figure 2.1: Simplified visualization of an ANN, showing inputs flowing through neurons that are all connected, resulting in an output [19].

The number of neurons present in a hidden layer and the quantity of hidden layers within ANNs exhibit significant variations. Certain ANNs consist of a single hidden layer with tens of neurons, while others have multiple hidden layers comprising of hundreds of neurons, depending on the complexity of the task and the desired performance of the NN. The purpose of the hidden layers is to identify the relevant features of the input data that influence the output. Each neuron in the current hidden layer is connected to all neurons from the previous layer, and the neural networks attempt to predict the output. The ANNs results are then compared to the actual output, generating a loss function. The neural network then seeks to minimize this loss by adjusting the weights of the neurons, often using a gradient descent optimization algorithm [20]. It is worth noting that recently, there are many optimization algorithms for ANNs. Among the most popular are SGD, ADAM [21], RMSprop [22] and AdaGrad [23], as well as variations of these.

One significant issue with ANNs is overfitting, which occurs when the network becomes too specialized to the training data and performs poorly on test data or the real data you seek to employ it for.

> "*Overfitting occurs when the gap between the training error and test error is too large.*" (Goodfellow et al., 2016, p. 110) [2]

Various techniques can be used to reduce overfitting, such as 2.1.5 data augmentation and early stopping [24]. Early stopping means halting training when the network's performance is deemed adequate.

Another slightly less common issue is underfitting, which happens when an ANN is not able to recognize patterns in the training data.

> "*Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set.*" (Goodfellow et al., 2016, p. 109) [2]

To combat this, one can introduce more neuron layers to make a deeper and more complex ANN. Another way to combat underfitting is by expanding the size of the training dataset, or to utilize 2.1.6 transfer learning.

2.1.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are an important part of computer vision models. A CNN is a type of an ANN that seeks to extract features from images through convolution and pooling layers [13]. Usually there are multiple convolutional and pooling layers, which produce the full scale model. A simplified CNN is visualized in Figure 2.2.

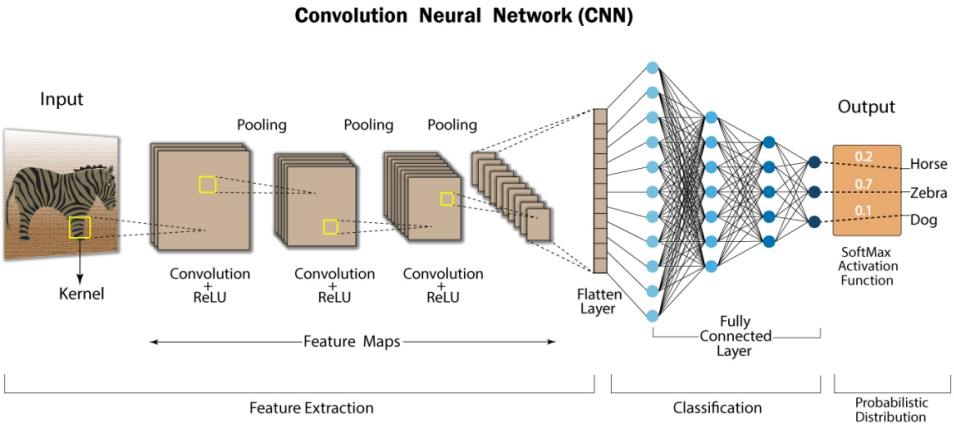


Figure 2.2: Simplified visualization of a CNN, showing how an image gets scanned and downsampled for feature extraction, before it is flattened and passed through a fully connected layer, resulting in a probability distribution of what animal the network thinks it is looking at [25].

Convolution

Convolution operates by applying a filter to an input image, where the filter is superimposed onto the input image and systematically shifted to generate a feature map. Figure 2.3 provides a simplified representation of this process. The filter encompasses various adjustable parameters, including kernel size, padding, and stride. Convolutional neural networks employ multiple filters concurrently on an input image. The amount of filters varies greatly. However, it is not unusual to use anything between 32 and 512 filters on a single input image.

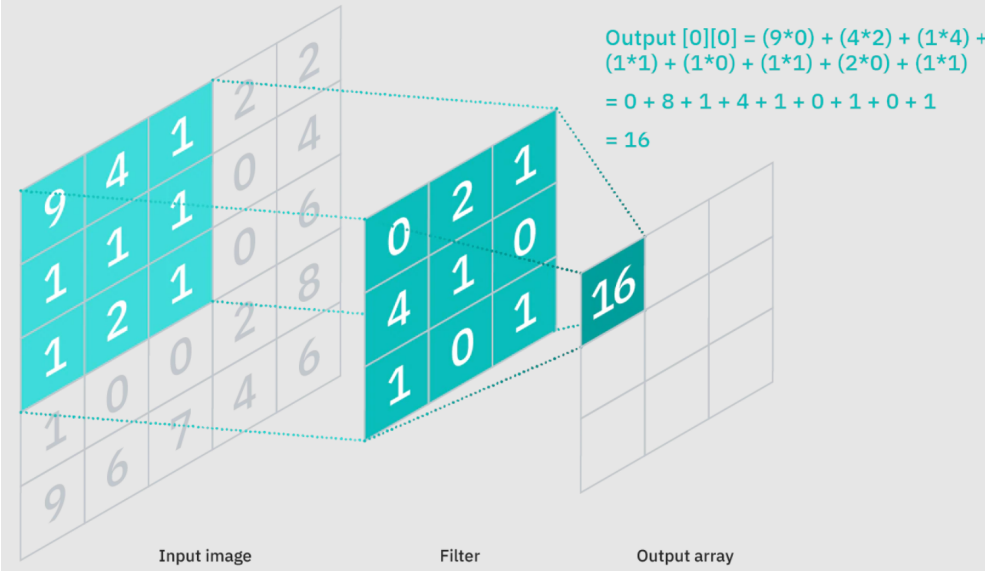


Figure 2.3: A simplified representation of convolution [26]

Pooling

The CNN then processes the convolved feature with a pooling layer. This is used to reduce the feature map [13], such that further convolution layers will compute faster. There are multiple techniques for pooling, with the most common being average pooling, and max pooling. These are visualized in Figure 2.4 and Figure 2.5 respectively. Average pooling works by computing the average of the convolved feature, to reduce a convolved feature from a n x n matrix to a single number. These numbers are combined to create a new, smaller feature map. This process is often repeated multiple times, as most CNNs have multiple convolution and pooling layers.

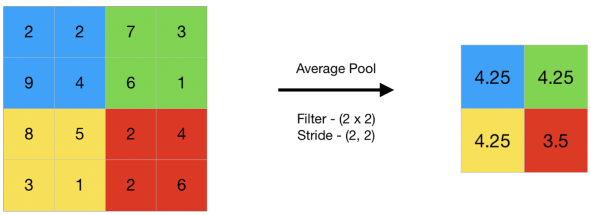


Figure 2.4: Average pooling with a 2x2 input and a number output [27]

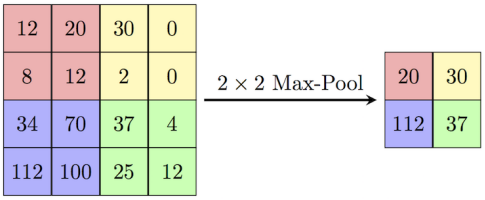


Figure 2.5: Max pooling visualization with a 2x2 input and a number output [28]

Stride

Stride refers to the number of steps which the kernel shifts across the input image during the convolutional operation. A larger stride value corresponds to a greater spatial displacement of the kernel, resulting in a reduced feature map size. A smaller stride value leads to a more fine-grained analysis with a larger feature map. This is illustrated in Figure 2.6.

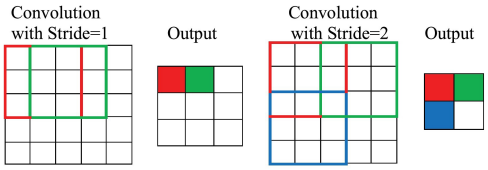


Figure 2.6: Visualization of 3x3 kernel(filter) with stride=1 and stride=2 [27]

Padding

Padding refers to the procedure of augmenting the input image with one or more layers of zero values, as depicted in Figure 2.7. This process is done to ensure the dimensions of the output feature map are adjusted appropriately during the convolution process [2].

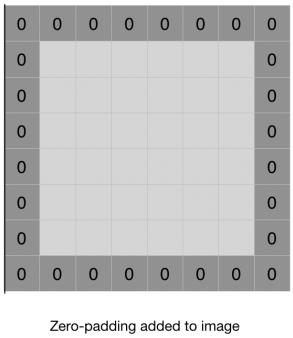


Figure 2.7: Image with one layer of zero-padding added [27]

Activation functions

CNNs and other types of NNs utilize activation functions to be able to introduce non-linearity to their calculations. This allows them to learn complex tasks, providing more advanced models. One such activation function often used, is the rectified linear unit (ReLU) activation function, first introduced in 1969 [29]. The ReLU is often used because it introduces non-linearity, and because it is simple and efficient. Given a variable x, the ReLU function is defined as:

$$\text{ReLU}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

2.1.5 Data augmentation

Data augmentation is a technique commonly employed to reduce overfitting, by artificially expanding the number of samples in the dataset by providing slightly adjusted copies of the original data [30]. Some examples of data augmentation can be rotating or adding blur to images. A model will generally perform better on new unseen data as it becomes more generalized due to more variations in the data set. Data augmentation is especially useful when training on smaller datasets, because they lack the quantity and diversity necessary for effectively training a model.

2.1.6 Transfer learning

Transfer learning refers to the concept of training a model for one specific problem, and then applying it to a different, but related, problem. One such example would be training a model to recognize dogs, and then employ said model to attempt to recognize cats. The concept of transfer learning has shown that it can greatly increase model efficiency by reusing and transferring the knowledge from one problem to similar one [31].

A number of models, such as ResNet [32], are often provided with weights that are pretrained on the ImageNet dataset [33]. Due to the large amount of images and variations of the dataset, it provides a rich source of visual information which can often be applied. By utilizing this technique, developers do not need to train models all the way from scratch, which may cause it to reach a better performance earlier.

In the context of object detection, models are often pretrained on the MS COCO dataset [34–36] to provide the model with a base of object detection. Some object detection models, such as SSD [36], contain a backbone to extract features, which are often pretrained on the ImageNet dataset.

2.1.7 Tensors

Tensors are data structures which are used to represent and manipulate multi-dimensional arrays of values. They can be seen as extensions to arrays and matrices, allowing for

data to be represented in various dimensions. Scalars can be seen as 0-dimensional tensors, arrays as 1-dimensional, matrices as 2-dimensional and so on. Tensors are highly utilized in the field of machine learning, as they are exceptionally flexible, and allow efficient computations on GPUs.

2.2 Image processing

2.2.1 Image layout

Image layout refers to the arrangement of pixels within an image. The layout determines the spatial arrangement of pixels, which influences the interpretation of the underlying data. Two commonly used layouts are the HWC (Height, Width, Channel) and CHW (Channel, Height, Width) formats. The channels refer to the color channels, such as RGB (Red, Green, Blue). A visualization of these two formats are shown in Figure 2.8.

Given a 1-dimensional array, which represents an image img, to find the pixel value at a given X and Y coordinate and channel C, the following indexing functions can be used:

- CHW: $(C \cdot img_{height} + Y) \cdot img_{width} + X$
- HWC: $(Y \cdot img_{width} + X) \cdot img_{channels} + C$

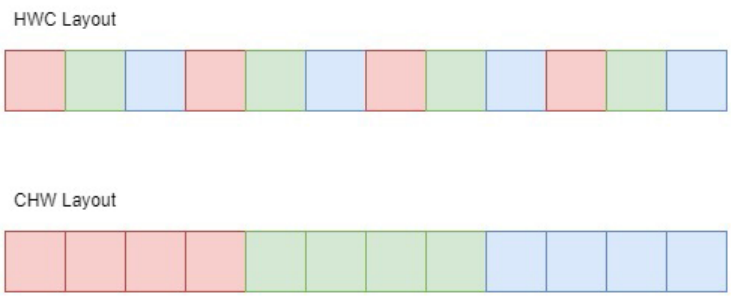


Figure 2.8: Visualization of RGB data with different layouts. The top layout uses HWC, and the bottom uses CHW.

2.2.2 Resizing

Resizing is an operation that involves changing the dimensions of an image. It is commonly used to adjust the size of an image for various purposes, such as display, printing, storage, or analysis. When resizing, interpolation methods are used to estimate pixel values based on the original image. The simplest method is nearest neighbour, which sets the value of the new image to the nearest pixel in the original image.

2.3 Evaluation metrics

2.3.1 True positive

A true positive (TP) is a result from an evaluation that correctly identifies a label.

2.3.2 False positive

A false positive (FP) is a result from an evaluation that incorrectly identifies a label.

2.3.3 True negative

A true negative (TN) is a result from an evaluation that correctly identifies the absence of a label.

2.3.4 False negative

A false negative (FN) is a result from an evaluation that incorrectly identifies the absence of a label.

2.3.5 Precision

Precision is a common metric within machine learning to determine the correctness of positives, and can be defined as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Precision measures the rate of true positives among all the positive predictions, and gives a score between 0.0 and 1.0, where a higher score is better and means that more of the positive predictions are accurate predictions.

2.3.6 Recall

In machine learning, recall is a measurement of the true positives correctly determined by the model, defined as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

In the context of object detection, a high recall score indicates that the model correctly identifies a large proportion of the objects that should be detected, while a lower score

indicates that the model is missing some or all of the relevant instances, resulting in false negatives.

2.3.7 Accuracy

Accuracy is a known metric within classification, which is the number of correct predictions, divided by the number of total predictions. Accuracy is a number between 0.0 and 1.0, with 0.0 being the worst, and 1.0 being the best. The mathematical formula for accuracy is as follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

2.3.8 IoU

Intersection over Union (IoU), also known as the Jaccard index, is a metric which describes the similarity of two sets [37]. The larger the value, the more similar they are. In the context of this thesis, the sets are rectangles. When comparing two sets, the IoU is evaluated to a value between 1.0 and 0.0. It is derived by dividing the area of the intersecting region by the area of the union of said rectangles. The IoU function is defined as follows:

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

An IoU of 1.0 means that the shapes completely overlap each other, meaning they are exactly the same shape and in the same location. Similarly, an IoU of 0 means that there is no overlap between the two areas. A sample of IoU scores are visualized in Figure 2.9.

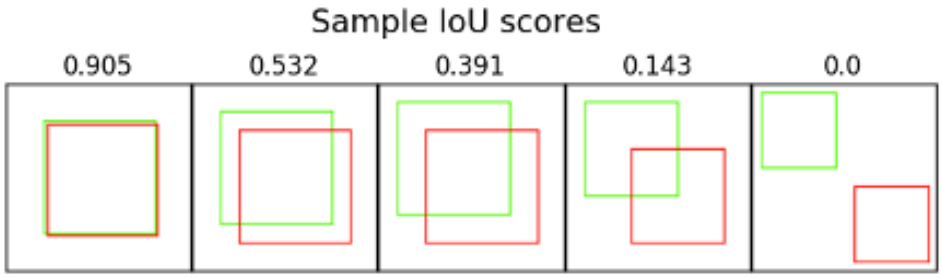


Figure 2.9: Various IoU scores with their respective visual overlaps [38].

This metric is widely employed in the field of computer vision, as it provides a straightforward means of identifying true positives, true negatives, false positives, and false negatives. This can then be used to determine the accuracy of a model. Additionally, it can be used in object tracking by determining if an object is located at the same position over time in a video.

2.3.9 AP

Average Precision (AP) is the area under the precision/recall curve, and is calculated by doing an integral on the precision/recall curve. This is illustrated in Figure 2.10.

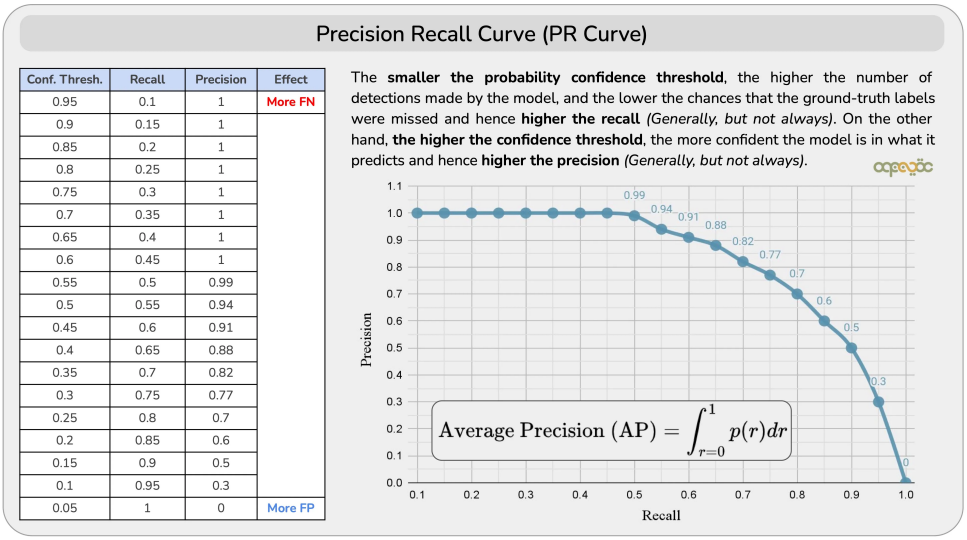


Figure 2.10: An example of a PR curve which is used to calculate AP [39]

The precision/recall curve is created by varying the classification threshold and calculating the corresponding precision and recall values. The precision/recall curve illustrates

the trade-off between precision and recall at different thresholds. The AP value ranges between 0 and 1, where a higher value indicates better performance. An example of how AP is calculated, is shown in Figure 2.11

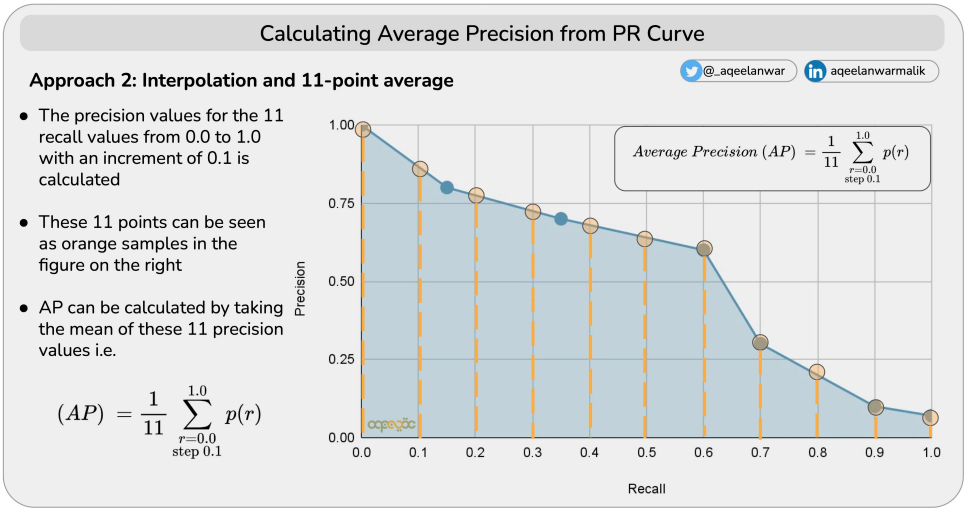


Figure 2.11: An example of how AP is calculated [39]

The mathematical formula for AP is:

$$\text{AP} = \int_{r=0}^{1} p(r)\,dr$$

2.3.10 mAP

Mean Average Precision (mAP) is a metric to determine how well a computer vision model performs. AP values can be calculated for each class and can then be used to calculate Mean Average Precision (mAP) for the whole model. mAP is the mean of AP across all the classes. As AP scores are between 0 and 1, mAP is also a value between 0 and 1, where 0 is least accurate, and 1 is most accurate. This metric is highly regarded due to incorporating both precision and recall, enabling a evaluation of the model's completeness across all classes.

The mAP metric is often specified with an accompanying value, such as mAP@50 or mAP@75. This numerical value defines the threshold for classifying an object as detected, based on the IoU. For instance, mAP@50 indicates that the AP algorithm uses an IoU threshold of 0.5 to determine if an object has been successfully detected. There is also a mAP@[.5:0.95] value, which is the average of the mAP values over the range of IoU's from 0.5 through 0.95, step 0.5, as can be seen in COCO source code [40].

The formula for calculating the mAP is presented as follows:

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} AP_i$$

In this equation, the variable N represents the total number of classes to which an object can be assigned.

In COCO context, AP is defined as the same as mAP [41].

2.4 Object detection

Object detection, an important aspect of computer vision, revolves around the identification and localization of specific objects within images or videos. It enables machine learning models to recognize and classify objects with a bounding box and label. This technology has extensive applications in various areas, such as Tesla's self-driving cars, surveillance [42], and wildlife protection [43].

Deep learning techniques, particularly CNNs, are used to increase the performance of object detection algorithms. CNNs can be used to learn certain patterns. When layers of these learned patterns are found and learned, they can be used to effectively deal with image data.

Object detection algorithms vary in the techniques they use. One architecture known as Region Based Convolutional Neural Networks (R-CNN) uses region-based methods, where a set amount of potential regions are selected, and tries to classify objects within those regions [34]. The other primary approach is called single-shot. Methods like these try to predict objects directly in one pass of the input image. These techniques are faster, but can sacrifice some accuracy. Some examples are Single Shot MultiBox Detector (SSD) [36] and You Only Look Once (YOLO) [35].

2.4.1 Non Maximum Suppression

Non Maximum Suppression (NMS) is a technique used in computer vision to select a single entity, such as bounding boxes, out of many overlapping entities. After sorting all input bounding boxes, the algorithm proceeds to choose the boxes with the highest confidence levels. However, if their Intersection over Union (IoU) score surpasses a predetermined threshold when compared to another previously selected bounding box, they are disregarded. It then repeats this until every box has been inspected, leaving only the remaining selected bounding boxes [44].

2.4.2 SSD

Single Shot MultiBox Detector (SSD) is an object detection architecture that consists of two primary components, namely a backbone and a detection head [36]. The backbone

typically refers to a pre-trained image classification network, such as ResNet, wherein the final fully connected layer is omitted. The SSD head component has the responsibility of classifying the detections. The architecture of SSD model is illustrated in Figure 2.12.

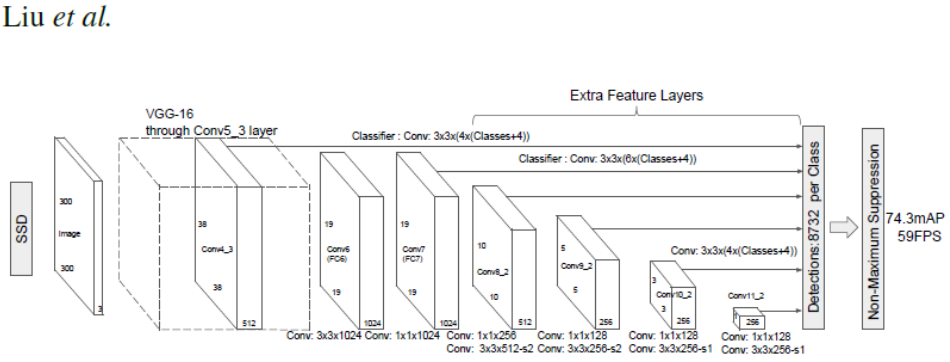


Figure 2.12: A visual representation of how SSD works [36]

Single Shot MultiBox Detector (SSD) works by dividing an image into a grid, and then having each grid cell responsible for finding objects within its boundaries. Grid cells that fail to identify any objects treat themselves as background regions of no interest and are consequently disregarded. Grid cells that successfully identify objects determine the corresponding region using the anchor box with the highest IoU.

Given that an object can potentially span multiple grid cells, the SSD model compensates for this by adjusting the scale of the anchor boxes. In addition, the SSD architecture incorporates multiple grid matrices to handle objects of varying sizes. For instance, it may initially employ a smaller grid, such as an 8x8 feature map, to detect and localize smaller objects, followed by a larger grid, such as a 2x2 feature map, to identify and locate larger objects.

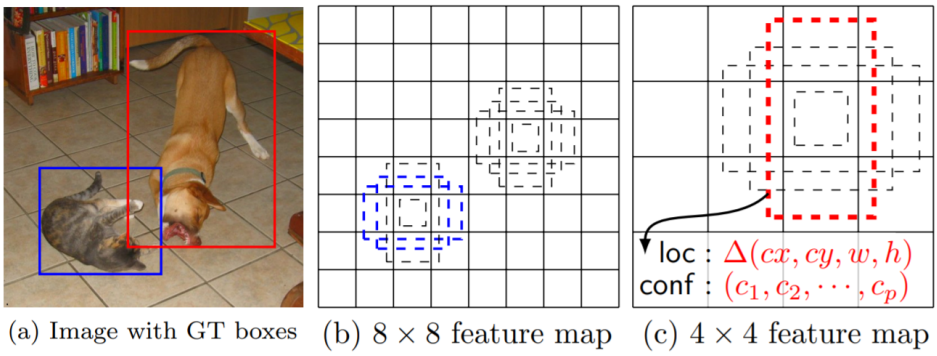


Figure 2.13: A visual representation of how the anchor boxes within SSD works [36]

In Figure 2.13, one can see the predetermined anchor boxes, and how the model selects the anchor box that overlaps the most with the given object. One can also see that the architecture has used different grid sizes/feature map sizes to detect different sized objects.

2.4.3 YOLO

The YOLO algorithm consists of a single Neural Network (NN) grid which takes an input image and outputs a set of bounding boxes and class probabilities for the objects detected in the image [45]. The network is composed of two main parts: a feature extractor and a detection head.

The feature extractor is a CNN that processes the input image and extracts a set of high-level features. These features are then fed into the detection head, which predicts the bounding boxes and class probabilities for the objects in the image.

To predict the bounding boxes, the detection head divides the input image into a grid of cells. Each cell is responsible for predicting a set of bounding boxes the grid have their centers inside the cell. For each bounding box, the detection head predicts the center coordinates, width, height, and confidence score.

The confidence score indicates how likely it is that the bounding box contains an object. This score is based on the IoU between the predicted box and the ground truth box. If the IoU is above a certain threshold, the box is considered to contain an object.

In addition to the bounding boxes, the detection head also predicts the class probabilities for each cell. These probabilities indicate the likelihood that an object of a particular class is present in the cell. The number of classes that can be detected depends on the dataset used for training the model.

Once the bounding boxes and class probabilities have been predicted for each cell, the

final set of detections is obtained by applying NMS to the set of predicted boxes.

YOLOv5

YOLOv5, released in 2020 by Ultralytics, is a highly efficient object detection algorithm that employs a deep neural network composed of a CSPNet-based feature extractor [46]. This feature extractor enhances YOLOv5's ability to extract more detailed features from input images, leading to improved object detection accuracy. The algorithm also utilizes a dynamic scaling approach and a multi-scale prediction strategy to detect objects of varying sizes. With the capability to detect up to 80 object categories, YOLOv5 outperforms previous versions and achieves state-of-the-art performance on the COCO dataset.

YOLOv5 offers five versions with varying model sizes. The smallest of which is YOLOv5n (1.9 million model parameters), to YOLOv5x (86.7 million model parameters). The larger models provide better mAP scores, at the expense of time required to run inference.

YOLOv7

YOLOv7 is an object detection algorithm that builds upon previous successes [45]. It includes small improvements to speed up inference and prioritizes accuracy during training without sacrificing speed. With models available in different sizes, YOLOv7 offers flexibility in balancing accuracy and computational efficiency.

YOLOv8

YOLOv8 is another object detection algorithm developed by Ultralytics, which released in January 2023 [47]. It features a CNN that consists of a module that handles feature extraction, referred to as a feature extractor, and a detection head. The latter predicts the bounding boxes and class probabilities for detected objects. YOLOv8 enhances efficiency by reducing box predictions and accelerating NMS. It introduces a modified loss function calculation compared to YOLOv5. Additionally, during training, YOLOv8 employs mosaic augmentation, except for the last ten epochs when it is disabled.

Like YOLOv5, YOLOv8 provides different versions with varying model sizes, ranging from YOLOv8n (3.2 million model parameters), to YOLOv8x (68.2 million model parameters).

2.4.4 Faster R-CNN

Faster R-CNN is an advanced object detection architecture built upon the previous R-CNN [48] and Fast R-CNN [49] networks. This model is designed to identify objects within an image and locate them using bounding boxes. The model operates by first proposing region of interests (ROIs) using a Region Proposal Network (RPN) [50], followed by classifying and refining these regions by using a Fast R-CNN network [49].

The RPN generates proposals by sliding a small network over the convolutional feature map of an image, and then scores the proposals based on their objectness. These proposals are then refined and classified using a Fast R-CNN network [49]. This approach allows for much faster object detection compared to the previous R-CNN and Fast R-CNN models, as the RPN can share convolutional features with the Fast R-CNN network [50]. This results in a more efficient and accurate object detection system.

2.5 HaGRID

HaGRID (Hand Gesture Recognition Image Dataset) is a dataset that contains 716 GB of hand gesture images with corresponding annotations [51]. HaGRID is split into 18 different gestures, and a `"no_hand_gesture"` class. The `"no_hand_gesture"` class is used when there is a hand in the picture that does not belong to the other gestures. The gestures contained within the dataset is visualized in Figure 2.14

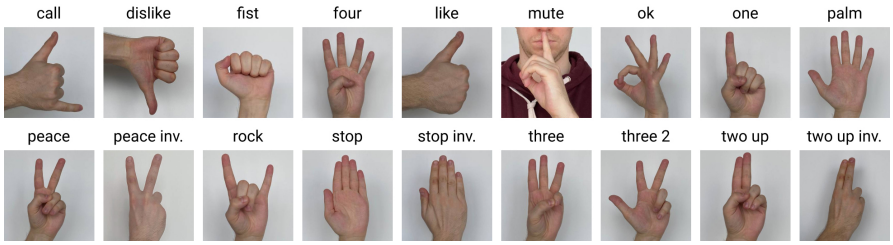


Figure 2.14: The gestures contained within the HaGRID dataset [51]

HaGRID's images are taken in various lighting conditions and with different backgrounds. In addition, the subjects' hand gestures are photographed from different distances between 0.5 meters to 4 meters away from the camera [51]. In Figure 2.15 one can see some images from the dataset with poor lighting conditions.

Figure 2.15: Some images with poor lighting conditions from the HaGRID dataset [51]

2.6 Data Formatting

Object detection models need specific inputs and outputs. Different models have different formatting on these values. There is a plethora of different dataset formatting for keypoints, landmarks and other points of interest. Most interest lies in bounding boxes as they both identify the object and provide them with a specific location in each image. During the investigation of the different models, a lot of different data formatting was used. A description of each of them are given below.

2.6.1 YOLO Formatting

You Only Look Once (YOLO) formatting is widely used for object detection tasks in computer vision. In YOLO, an image is divided into a grid, and each grid cell is responsible for predicting bounding boxes for objects within its region. The bounding box is represented by four values: the coordinates of the top-left corner, the width, and the height. Additionally, each bounding box is assigned a class label, indicating the object category it represents [52]. For example, in the YOLO format, a bounding box for a car in an image may be represented as follows:

```
car 0.65 0.32 0.20 0.45
```

Here, "car" denotes the object class, while the subsequent four values represent the coordinates and dimensions of the bounding box.

2.6.2 COCO formatting

Common Objects in Context (COCO) formatting is widely adopted for various computer vision tasks such as object detection, instance segmentation, and keypoint detection. COCO utilizes a JavaScript Object Notation (JSON) file to store the annotations for each image. The annotations include information such as bounding box coordinates, segmentation masks, and category labels [53]. For example, a COCO annotation for an object detection task may look like this:

```
{
    "image_id": 123,
    "category_id": 2,
    "bbox": [100, 200, 150, 100]
}
```

In this example, `image_id` refers to the unique identifier of the image, `category_id` represents the class label, and `bbox` denotes the bounding box coordinates in the format [x, y, width, height].

2.6.3 PASCAL VOC Formatting

Pascal Visual Object Classes (VOC) formatting is commonly used for object detection and segmentation tasks. It employs eXtensible Markup Language (XML) files to store annotations for each image. The annotations include bounding box coordinates, object labels, and flags for difficult or truncated objects [53]. For instance, a Pascal bounding box annotation for a bounding box may look like this:

```
<object>
   <name>car</name>
   <bndbox>
      <xmin>100</xmin>
      <ymin>200</ymin>
      <xmax>250</xmax>
      <ymax>300</ymax>
   </bndbox>
</object>
```

In this example, "name" represents the object class, while "xmin," "ymin," "xmax," and "ymax" denote the coordinates of the bounding box.

2.6.4 TensorRecords

TensorRecords is a labeling and data serialization format designed for efficient storage and processing of large-scale datasets. It is used in the popular machine learning framework TensorFlow. TensorRecords provide a unified format to organize diverse data types such as images, audio, and text. The format allows for efficient streaming and shuffling of data, making it suitable for training large-scale models [54].

2.7 Technologies

2.7.1 Roboflow

Roboflow is a computer vision platform that has a labeling tool which is used to annotate and label objects in a an image [55]. The tool is equipped with an integrated feature that enables automatic labeling of images. This makes manual labeling a lot faster, as one only has to add missing annotations or make minor adjustments to existing annotations. After the images are labeled, one can download the dataset with labeling in different formats, such as, but not limited to: yolo, coco and tensorflow format.

2.7.2 Git and GitHub

Git is a version control system created in 2005. It tracks changes in files, which allows developers to manage code [56]. Git is often used in collaborative projects with multiple developers. Git accomplishes this by making snapshots of the files, storing them, and referecing them. Git also provides functionality to merge, branch out, rebase, pull and push changes to projects.

GitHub is a web-based platform that utilizes Git, and provides a centralized location for code repositories, making it easier for developers to manage their projects [57]. GitHub also provides their own features, such as issue tracking, automated testing and deployment, which streamlines the software development process.

2.7.3 Docker

Docker is an open-source platform, which facilitates development, deployment, and management of applications through containers [58]. Containers are lightweight and portable images, which encapsulate an application and its dependencies. This ensures that execution is consistent regardless of deployment environment. Docker consists of the Docker Engine, a core component responsible for building and running containers, and Docker Hub, a cloud-based registry for storing and sharing container images. By isolating applications in containers, Docker streamlines the continuous integration (CI) and continuous deployment (CD) process, and encourages a more efficient and reliable approach to application management.

2.7.4 C and C++

C is a general-purpose programming language. It provides low-level access to system memory, thus making it exceedingly fast. It does not provide garbage collection, and expects the developer to explicitly allocate and deallocate memory. C++ is an object-oriented programming language, which makes it convenient to structure applications. C++ was developed as an extension of C, thus, it shares a lot of the same syntax as C. The main difference between the two is that C++ deals with objects and classes, while C does not support this [59] [60].

2.7.5 GStreamer

GStreamer is an open-source multimedia framework which enables development of applications which can process, create, and manipulate multimedia content, such as audio or video [61]. The framework is designed to facilitate a modular and pipeline-based approach. GStreamer is written in C. GStreamer provides a vast amount of plugins which can be downloaded and used, such as scaling images, or grab video from webcams. A pipeline can be launched in the command line using `gst-launch-1.0`, followed by any GStreamer elements, which are the basic building blocks of any pipeline. However, using many GStreamer elements to dynamically solve different tasks can be challenging and error-prone. To address this issue, the concept of "bins" [62] can be implemented. Bins are containers that can contain other elements. The use of bins enables the handling of complex architectures and the testing of a subset of functionality to ensure that it performs correctly. To make GStreamer elements work, they utilize "pads", which are their interfaces to the outside world. The pads mainly describe their direction, which is either "sink" or "source". Sink pads bring data inside an element, and source pads send data out of an element.

2.7.6 OpenVINO

OpenVINO is an open-source toolkit developed to facilitate deep learning model optimization and deployment [63]. It provides a large suite of tools and libraries for high performance model inference. Some relevant tools provided, are the OpenVINO Runtime and Model Optimizer. The OpenVINO Runtime is responsible for loading the provided models and executing inference requests within said models. The Model Optimizer is a tool which aims to convert ONNX models from `.onnx` files, to an Intermediate Representation (IR), consisting of a `.bin`- and `.xml` file, which the OpenVINO Runtime can efficiently process.

2.7.7 Python

Python is a high-level, interpreted, general-purpose programming language. It focuses on easy to read- and write code. It has an extensive standard library, which makes it simple to make all kinds of applications. In addition, one can use libraries made by other developers by utilizing pip [64], which is a package manager for Python [65].

2.7.8 PyTorch and TensorFlow

PyTorch and TensorFlow are open-source Python frameworks for developing and training machine learning models and doing tensor math. They have well documented APIs, which makes it easy to develop and train models. The two frameworks make up a huge portion of used technologies used in machine learning projects, research, and ecosystem [66] [67].

2.7.9 ONNX

Open Neural Network Exchange (ONNX), is an open-source platform that enables interoperability between machine learning frameworks [68]. ONNX aims to be a standardized representation for trained models from different frameworks, such as PyTorch and TensorFlow. It allows those models to be seamlessly converted and utilized across platforms. By providing a common format for the representation of machine learning models, ONNX provides optimization of model performance across hardware platforms and makes the deployment process easy. With the standardized model representation, a file format `.onnx` is used. ONNX defines a common set of operators, and the `.onnx` file format utilizes it. The operators are versioned, with their versions being referred to as operator sets.

2.7.10 Cloud GPU clusters

A cloud GPU cluster is a cluster of GPUs hosted in an online environment, which facilitates the ability for users to utilize the resources they provide, such that they do not need to buy their own hardware. These clusters usually provide hardware that is exceptionally powerful compared to regular consumer hardware for GPU computing, and allows a user to utilize multiple of them. GPU clusters are, for that reason, very useful for training machine learning models. NTNU provides one such cluster to use for students, Idun [69]. Alternatively, multiple online services provide cloud computing as a service, and can be rented.

2.8 Test-driven development

Test-driven development (TDD) is a software development practice that revolves around creating unit tests before software functionality. TDD makes it easier to write robust, bug-free and maintainable code by continuously refactoring code to make it better, while constantly testing the code to make sure that it still works.

2.9 Pexip Codebase

The company's codebase is built using GStreamer, a multimedia framework that supports the construction of complex multimedia applications. The codebase consists of thousands of dynamically linked elements, with each element performing a specific functional task.

2.9.1 Mama

Mama is one of the main bins in the Pexip codebase that manages audio and video processing in a complex system. Mama is the highest level of bin, containing hundreds of other GStreamer elements in different hierarchies, and it manages all audio and video processing in the media stack.

2.9.2 PMX

PMX is a C API that provides a high-level interface with inputs and outputs of different types that can be connected together. PMX abstracts away the details of GStreamer and configures Mama, which have complex APIs, to offer simpler functionality. In essence, PMX simplifies the configuration process for Mama and other large bins, by providing an easier-to-use interface built on top of their complex APIs.

2.9.3 Web Application

The web application refers to the overarching software that makes it possible to participate in video conferences. The web application utilizes the PMX API for various functionalities and purposes.

The web application provides a server which users can connect to. There are various types of users that can join, which have different capabilities, and are treated differently. For example, a user who participates through a link on a browser will be different from a user who participates through a Cisco touchpad.

Chapter 3

Methodology

The objective was to develop an efficient gesture detection system for video calls. A suitable dataset encompassing relevant gestures for video call interactions was chosen. The dataset was utilized to train various object detection models. Subsequently, different models were evaluated, considering factors such as framerate and mAP scores.

Lastly, the implementation of the gesture detection model within the Pexip code base will be discussed. The models were integrated into the existing application, incorporating necessary modifications and implementing safeguards to ensure seamless operation. The specific steps taken to incorporate the models, adapt to application requirements, and optimize performance within the Pexip environment will be covered.

The utilization of the technologies mentioned in the theory section were necessary either due to required workflows by Pexip, such as Docker and the Pexip provided development image, or were essential to effectively produce other components. Not only one machine learning framework was utilized. The reason for not utilizing only one machine learning framework, was due to each implementation's ease of use. This also led to the usage of ONNX to create standardized representations between every framework.

3.1 Gesture Detection Model

The objective was to develop a gesture detection model that treated each gesture as a distinct object, characterized by a bounding box and a confidence score. To accomplish this, various models were explored that aligned with the requirements. The models considered included SSD, YOLOv5, YOLOv8, and Faster R-CNN. These models were chosen for their ability to capture and localize gestures effectively and accurately.

3.1.1 Dataset

For training data, the test set from HaGRID was utilized as the primary dataset. The test set had a considerable size of over 60.4 GB and included a diverse range of pictures with

different lighting conditions, ethnicities, and resolutions. The extensive variability found in the HaGRID test set made it an ideal choice for training purposes. The predefined subset of HaGRID, comprising 100 pictures for each class, was utilized as the test set. This subset comprised around 2.5 GB of data. Despite its relatively small size, the test set demonstrated a satisfactory level of diversity, encompassing various backgrounds, noise, and contextual variations in each picture. Therefore, this dataset was considered suitable and adequate for training the gesture detection model.

Figure 3.1: Composition of 10 images from HaGRID dataset [51]

3.1.2 Model selection

The selection of the models was based on a combination of factors, including ease of implementation, training efficiency, evaluation capability, final model size on disk, and inference speed. The models that were chosen were YOLOv5s, YOLOv8m, SSD ResNet50 V1 FPN 640x640 and Faster R-CNN ResNet50 V1 640x640 [67].

The YOLO models were chosen for their user-friendly documentation, state-of-the-art performance in terms of high mAP scores on the COCO dataset, and fast inference. The YOLOv5s exhibited fast inference times, but had lower mAP scores on the COCO dataset compared to YOLOv8m. Additionally, a pre-trained YOLOv7 model was fetched from the HaGRID repository in the early stages, which allowed working on a base implementation while waiting for further models to be trained.

As this thesis revolves around detecting and distinguishing objects that are quite similar, we did not want to exclude the possibility that other models may have produced satisfactory results. We chose SSD and Faster R-CNN models because they demonstrated desirable precision scores and efficient inference speeds. Specifically, the SSD ResNet50 V1 FPN 640x640 achieved a mAP of 0.46 with an inference speed of 34.3 ms, while the

Faster R-CNN ResNet50 V1 640x640 achieved an mAP of 0.55 with an inference speed of 31.8 ms on the COCO dataset [67].

3.1.3 Formatting

HaGRID is formatted following the COCO format, which was incompatible with models that had been selected. However, HaGRID's GitHub repository [51] provided a convenient solution in the form of the `coco_toYolo.py` script, enabling the conversion of annotations to the YOLO format. This allowed training the two YOLO models that had been selected.

To utilize the SSD and Faster R-CNN models, the input required TFRecord files. These models were sourced from TensorFlow's model zoo [70]. Initially, a script called `YOLO-convert-txt-2-xml` developed by the GitHub user chanjooLee [71] was used to convert the YOLO formatting to Pascal VOC formatting. Next, a script was created to convert this formatting into CSV format. Finally, a modified version of the `generate tfrecord.py` code from the TensorFlow Object Detection API, originally modified by Dat Tran [72], was employed to generate the necessary TFRecord files.

This multi-stage formatting process allowed us to create a `train.record` file containing all the training data, as well as a `test.record` file with the test data. These TFRecord files ensured compatibility with the SSD and Faster R-CNN models, facilitating the training and evaluation processes effectively.

3.1.4 Model training

The training of the YOLO models took place on the Idun GPU cluster provided by NTNU.

The YOLOv5 model training involved utilizing the YOLOv5 repository from ultralytics [46]. The training process consisted of executing the training script `training.py` and adjusting parameters. The batch size was adjusted to 30 and the image size to 480. The initial plan was to train the model for 200 epochs. However, the training ended prematurely at 96 epochs due to the lack of significant improvement in the mAP scores over a considerable number of epochs. This training process took approximately 4 days.

The training process for the YOLOv8 model followed a similar approach. The ultralytics pip package [73] was utilized, and the train command was employed. Initially, the training was set to run for 100 epochs. However, it was decided to conclude it at epoch 57 due to the absence of significant improvement in the mAP score beyond this point. The YOLOv8 model underwent training with an image size of 640 and a batch size of 8. Notably, the training of this model took approximately 6 days to complete.

During the training process, the YOLO models conducted real-time evaluation and generated a `result.csv` file. The metrics included in this file were utilized to assess the

performance of the model.

The SSD model was trained using the TensorFlow (TF) Object Detection API. The training was done by using their `model_main_tf2.py` script. To configure the training, we specified the batch size, step number, number of classes,label map, as well as the paths to the training and test data in a configuration file [74]. The model was trained for a total of 15,000 steps.

Similarly, the Faster R-CNN model training followed a similar process using the TensorFlow Object Detection API [74]. The training process was halted at 9,000 steps.

The evaluation of TF models took place after completing their training. This approach involved running the training script again, but with specific flags indicating the intention to evaluate a particular model. The mAP scores for these models were measured at the end of their training, using the evaluation script. Real-time monitoring of the total loss of the model was utilized to determine when to halt the training process.

For detailed configuration files used in these trainings, please refer to the code listings in appendix B.

3.1.5 Exporting to ONNX

To be able to make further use of the models, they were required to be `.onnx` files. The implementations of each architecture had their own export scripts, which could be run with Python. The scripts expected to find completely trained models which they could export. The scripts allows users to define the operator set versions of the output file. While exporting, the operator set version was set to 13.

3.2 GStreamer element

To be able to utilize trained models with Mama and PMX, a GStreamer element must be created, and then plugged into Mama. Said element must be responsible for emitting signals with labels that correspond to different gestures that are identified throughout its lifetime. It has to load and initialize a specified and compatible model, accept image data, transform it so that the model can use the data, run inference on the said data, and tell Mama and PMX when it has detected a gesture with high confidence. The full GStreamer element consists of the main element itself, an inference class to interact with the OpenVINO Runtime that runs the model, and some helper classes. Further in this section is the in-depth explanation of the full GStreamer element.

3.2.1 Gesture detection element

A C++ class was needed to contain various instances and variables. To let our class act as a GStreamer element, it must inherit from a GStreamer base class `GstVideoFilter`.

Then some boilerplate code is required. This includes some C and C++ definitions and macro definitions, a source and sink pad, initialization functions for the class itself and instantiation of the class, a dispose function, and a function which takes in image data. To get this done quickly, help was provided by Pexip. Further, *the element* will be used to refer to this GStreamer element.

To work comfortably in this setting, it is very helpful to have at least basic experience with C and C++ code.

The element must be defined with a number of class members which act as configurations. These configurations are responsible for giving the user a variety of final settings, allowing different use cases. These configurations must be tunable to simplify experimenting, without compiling new values each time. To accomplish this, some additional GStreamer functions must be added [75]. An enum must be created, with a value for every property one desires to be able to easily change. These enum values are then "installed" into the element class within the class initialization function, with minimum, maximum and default values. Further, two more functions must be created, to get and set the properties at any time. For our element, the configuration properties are listed below:

- `debug`: Enables drawing detection boxes on the screen.
- `debug_overlay`: Enables drawing an overlay, which displays the data that is handed to the detector, after preprocessing.
- `confidence_threshold`: The minimum required value for a detection to not be discarded.
- `iou_threshold`: The minimum required IoU required for a compared detection to not be discarded.
- `min_detections`: The minimum amount of consecutive, similar detections required to emit a signal.

By utilizing the GStreamer object properties, a user can specify desired values for said properties through code, or adding values directly when running it in the command line.

For the element to be able to notify Mama and PMX that a gesture has been found, it must utilize GStreamer signals. An enum must be created, with a value which represents the event "a gesture has been detected". This signal, with the metadata it is expected to provide, is defined in the class initialization function.

When data is passed to the detection element, we want to process it. This data is an object which contains frame data from the video-stream. To process the current frame, it becomes necessary to extract the RGB pixel data, height, and width from the frame. When this is complete, said data is passed to a detection method defined on a separate

detector class. Since the frame data is structured as a 1-dimensional array of values, the height and width is required as additional context, because the 1-dimensional array alone cannot convey the height and width of the frame. Inference is run on the data by the detection class, and returns an array of gestures it found, which may be empty. Section 3.2.2 contains further details about the detection class. After inference, the results undergo a layer of postprocessing to ensure higher quality detections. This postprocessing consists of an algorithm designed to monitor gestures with the same label within a similar region of the frame. A signal is emitted only if a gesture has remained in the same area with the same label throughout a given amount frames in the video-stream. This amount corresponds to the `min_detections` property. The algorithm is explained and illustrated in 3.2.1.

Result classes

Representations for the detections were created, to be able to easily work with them. Some classes were created. The relationship between these classes are visualized in Figure 3.2.

Bounding boxes A class for bounding boxes. This class represents the X and Y coordinates of their top left corner, and the height and width of the box. Additionally, it also contains necessary methods to calculate the IoU between itself and another bounding box.

Result A class to represent a full detection. This consists of a bounding box, a confidence value, and a label. This class will be referred to as a `Result`.

Result with lifespan A helper class for the algorithm in 3.2.1, which class contains a `Result`, and an integer `lifespan`. The `lifespan` represents the number of consecutive detections in which a `Result` with the same label, and same general area has been detected. This class will be referred to as a `ResultWithLifespan` (RWL).

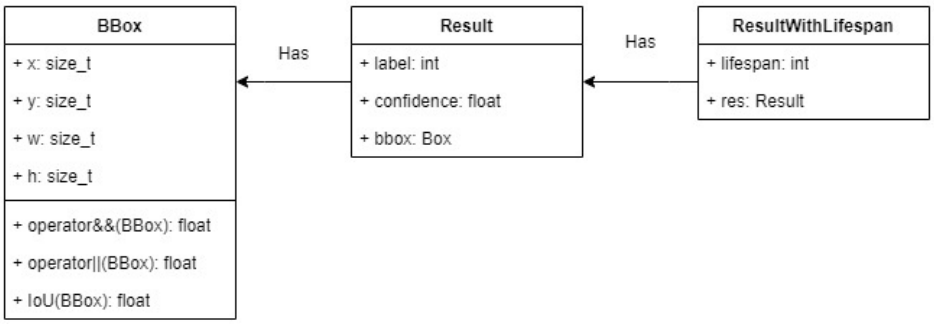


Figure 3.2: The three necessary result classes and their relationships.

Detection processing algorithm

To ensure high quality detections, an algorithm which attempts to track detections was made. This is to reduce the amount of false positives, such as a person walking in the background with their hands up, participants scratching their nose, or flailing their arms around. This can only be done through multiple iterations. Thus, it has to process detections across several frames, as the video-stream progresses. The pseudocode for this algorithm is shown in Algorithm 1. The algorithm uses the `iou_threshold`, `confidence_threshold`, and `min_detections` parameters in the GStreamer element. Additionally, it requires the element to track valid detections from the previous frame. These detections are stored as a vector of RWL.

The algorithm incorporates a vector of `Result` and inspects all of them. If the confidence values are below the threshold, they are discarded. Otherwise, they are compared against every RWL from the previous frame. During comparison, if the `Result` has the same label, and is above the `iou_threshold` as the RWL it is compared against, it is considered the RWL's successor. If this is the case, the `Result` that the RWL contains is replaced by the current `Result`, and it's lifespan is increased by 1. This effectively moves the RWL bounding box, such that a new `Result` is not compared against the very first bounding box for the RWL in question, but the most recent one. If the `Result` is not considered a successor, it will become a new RWL, with a lifespan set to 1.

During the comparisons, a check was made to examine if the lifespan of any RWL becomes equal to `min_detections`. If that is the case, a signal is emitted, containing the label of the detection. This is checked for every comparison, such that the algorithm can emit several signals for every frame, in case there are multiple gestures found. The algorithm is visualized with 3 frames, and `min_detections` set to 3, in Figure 3.3.

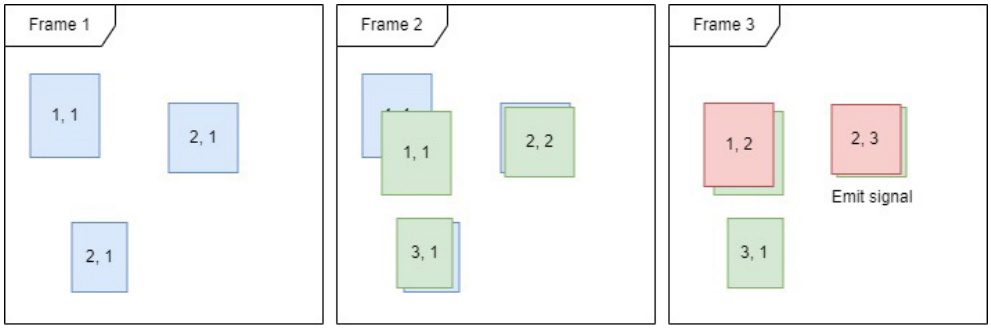


Figure 3.3: Visualization of processing algorithm. The boxes represent RWLs, at the end of the algorithm, formatted as **[LABEL, LIFESPAN]**. Frame 1: 3 new detections (blue). Frame 2: 3 new detections (green). Top left green is below the IoU threshold. Bottom left has a different label. Top right is above IoU threshold and has same label, thus, it's lifespan is increased to 2. Frame 3: 2 new detections. Top left is now within the IoU threshold and label. No new detection for bottom left. Top right is above IoU threshold and has same label, thus, it's lifespan is increased to 3. As it's lifespan reaches 3, a signal is emitted.

Algorithm 1 Temporal result detection algorithm

Require: Det_{new}: a vector of `Result`
Require: Det_{old}: a reference to a vector of RWL
Require: T_{IoU}: an IoU threshold
Require: T_{conf}: a confidence threshold
Require: M: minimum RWL lifespan for signal emission
Ensure: Temporal result detection

$tmp\_results \leftarrow$ empty RWL vector
for all $det \in Det_{new}$ **do**
  if $det.confidence < T_{conf}$ **then**
    continue
  end if
  $overlapping \leftarrow$ new RWL with $det.bbox$ and $lifespan = -1$
  for all $old\_det \in Det_{old}$ **do**
    if $det.label \neq old\_det.label$ **then**
      continue
    end if
    if $IoU(det, old\_det) < T_{IoU}$ **then**
      continue
    end if
    $overlapping \leftarrow$ new RWL with $old\_det.bbox$ and $old\_det.lifespan + 1$
  end for
  if $overlapping.lifespan == -1$ **then**
    $overlapping \leftarrow$ new RWL with $det.bbox$ and $lifespan = 1$
  end if
  $tmp\_results$.push($overlapping$)
  if $overlapping.lifespan == M$ **then**
    emit ($det.label$**)**
  end if
end for
$Det_{old} \leftarrow tmp\_results$

Debugging

For easier development and comprehension, two debugging features were added. First, drawing an overlay at the top left of the video-stream. This overlay displays the image data which is fed directly into the object detection model. It means the user can easily see the image data after preprocessing. E.g. resizing the image. To do this, an array is created with the same size as the array that the model itself accepts, and populated with the preprocessed data, and lastly drawn on top of the main video-stream.

Second, drawing boxes on top of the image. This allows users to see the bounding boxes of detections in real-time. This requires the `Result` for the detections. It draws the bounding box of each detection on top of the video-stream. The bounding boxes are

colored either red, yellow or green, based on the confidence level of the detection they represent. Given a confidence x, then the color is given by the function

$$f(x) = \begin{cases} \text{Green}, & \text{if } x > 0.9 \\ \text{Yellow}, & \text{if } x > 0.5 \\ \text{Red}, & \text{otherwise} \end{cases}$$

These two debugging features are visualized in Figure 3.4. Additionally, labels and confidence values can be logged to the console for each detection. This can be achieved by setting the debug level to DEBUG when launching the GStreamer pipeline.

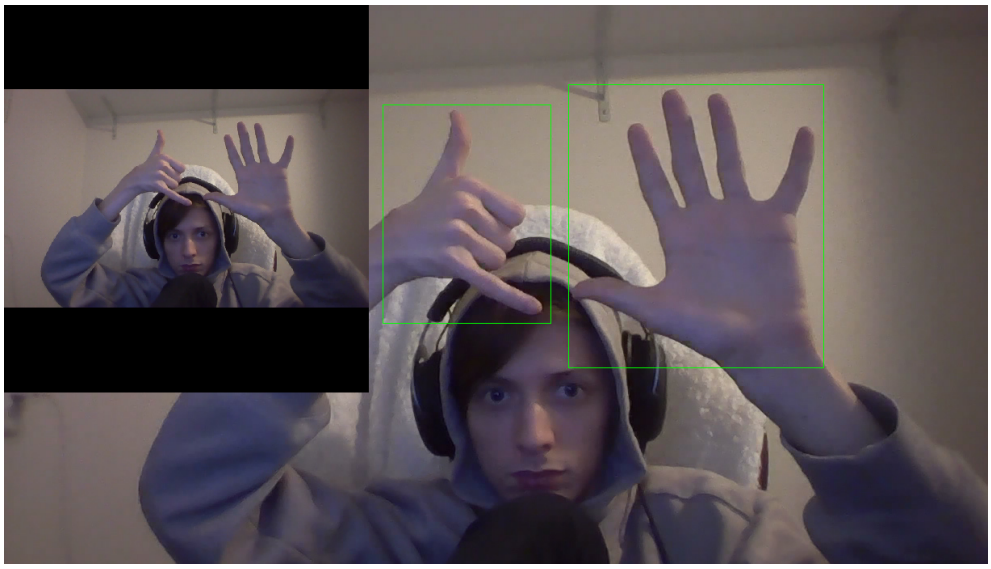


Figure 3.4: Visualization of the overlay and box drawing debug features.

3.2.2 Inference class

To separate the GStreamer functionality from the object detection functionality, another C++ class was created to handle the latter. It's purpose is to control all aspects of running a trained model with the OpenVINO Runtime, such as processing image input, running inference, transforming the raw output, and postprocessing.

The class must contain a number of OpenVINO InferenceEngine members, namely `InferenceEngine::Core`, `InferenceEngine::ExecutableNetwork`, and `InferenceEngine::CNNNetwork`. It also contains a `Model` enum, to make it explicit which model is being used, e.g. YOLOv5, YOLOv7. Lastly, it must also keep a confidence threshold and the input width and height which a loaded model accepts.

Several methods were also defined. These include the detection method itself, which branches out to private detection methods, which are used dependent on the selected

model, and a heap of preprocessing and postprocessing helper methods. The GStreamer element calls the detection function.

Model loading

The detector class initially loads an OpenVINO Runtime plugin `.xml`-file, which has to be located. This attempts to initialize a backend, and throws an exception if it cannot be initialized. If successful, it provides information about available devices, such as CPUs or GPUs, and their properties. Depending on which model is specified to be used, an absolute file path is set for each model file, which are `.onnx` files. The operator sets must be version 13, otherwise it does not load, and crashes. If the model successfully loads, it populates the rest of the class members, and grabs the input height and width that the model requires. The network that is loaded can then be further configured, allowing users to specify the image layout and floating point precision it should accept and output. It also allows preprocessing to be specified, but this does not work if OpenCV is not installed.

Image preprocessing

The loaded models usually accept data in a format that is different from what is produced from a video-stream. To handle this, preprocessing is necessary to transform the data to an accepted format. As mentioned in 3.2.1, the GStreamer element passes data to a detection method. The data consists of an array representing the image input, the height and width of the input image, and an empty array that may be populated at the end of the preprocessing, which is the data that allows the drawing of the debug overlay.

To make the input image data compatible with the model, it has to go through a series of transformations. Firstly, it must be resized to match the resolution of the images that the model was trained on. These values can be variable, but are provided by the `.onnx` files when a model is loaded. The resizing used was the nearest-neighbour interpolation algorithm. Further, an image cannot be simply resized. It must also keep its aspect ratio, such that it does not become distorted. A letterboxing effect is used to achieve this. The scaling values from resizing and letterboxing is stored to be able to reverse the down-scaled size of any detections bounding box. Lastly, the input image data will be in the HWC image layout, and must be converted to CHW.

To accomplish this, a loop is utilized to swap the layout by iterating through the data by their channels, height and width, and then indexing into an input and output array of the same size for both HWC and CHW respectively, then setting the pixel in the CHW layout to the corresponding pixel in the HWC layout.

When these modifications were done, the processed input images were unflattened to a 4-dimensional tensor which the model accepts. The dimensions of this tensor must be [1, C, H, W]. The 1 must be added as the model accepts a batch of several images, but in

this case, the batch size is 1. The H and W refers to the height and width that the model expects.

When preprocessing is complete, the processed image may look as the top left overlay shown earlier in Figure 3.4.

Model outputs and postprocessing

Different models have different implementations. Currently, only YOLOv5 and YOLOv7 are fully implemented. Even though they are both YOLO models, they have different outputs. This means they require different implementations. The outputs for both of these models are tensors consisting of multiple detections, but the underlying values are different.

For v5, only $\frac{1}{3}$ of the detections had to be examined due to anchor box calculations. Detections in v5 could have multiple labels, thus, each detection had a list of 24 values, 5 of them describing location and objectness, while remaining values describe the confidence values for each gesture label.

Detections in v7 had only 7 values, since it did not have multiple labels per detection. The consisted of objectness, location, confidence for the best label, and index of the best label. For v5, the label with the highest confidence was chosen and the rest were discarded. For both implementations, the values for each detection were collected and transformed into `Result` instances. Transformation was handled by a helper method, which required the previously mentioned scaling values from the resizing and letterboxing done in the preprocessing step. The values are used to inverse the downscaling, resizsing the bounding boxes such that they fit the full-size input image. This helper method required per-model implementations, as the X and Y coordinates of the bounding boxes produced by v5 represent the center coordinates, while the X and Y coordinates in v7 represent the top left corner.

The YOLOv5 model did not apply its own NMS, but v7 did. The v5 results were filtered through NMS to make them consistent.

3.3 Integration into Pexip codebase

With a complete GStreamer element, it was plugged into Mama and PMX. This had to be done to complete the several goals, such as joining a video conference and letting the users talk and raise their hands. To do this, supervision from Pexip was needed.

Firstly, the detection element was plugged into Mama, allowing gestures to be emitted from the detector to Mama. Secondly, the emission needed to be brought from Mama to PMX to allow PMX to be notified if a signal is emitted. As PMX allows functions to be called when certain events happen, aka. callbacks, the web application could inform

PMX functions it should call when a signal is emitted. This was the exact functionality that was used for the web application to trigger the "raised hand" functionality when a gesture was detected.

As the webapp allows different types of users to participate in a meeting, such as normal users or e.g. Cisco touchpads, some additional code had to be written to ensure that all types of participants would be able to utilize the detection element, and thus, the "raise hand" functionality. As the gesture detection element emits a label for any gesture it detects, it is important to filter out the labels that are unwanted. Thus, when a user raises their hand, it is not desired to use e.g. a "thumbs up" gesture to trigger the "raise hand" functionality. For that reason, only some gestures were selected to be capable of triggering the "raise hand" functionality:

- Stop
- Palm
- One finger
- Two fingers
- Four fingers
- Peace

For reference, these gestures are visualized in Figure 2.14. Additionally, for demo purposes, a "fist" gesture was added to allow a participant to use the "lower hand" functionality, which can be used after they have raised their hand.

3.4 Usage and performance

To run the GStreamer pipeline with the detection element, `gst-launch-1.0` pipelines were utilized for different usecases. To simply use the element and have it log when a signal is emitted, the command shown in code listing 3.1 was used. For testing and debugging, adding debug flags could be used, and is shown in code listing 3.2.

Code listing 3.1: Base gst-launch-1.0 pipeline to use the element.

```
$ gst-launch-1.0 v4l2src ! jpegdec ! gesturedetect ! videoconvert ! ximagesink
    async=0 sync=0
```

Code listing 3.2: gst-launch-1.0 pipeline utilized when debugging.

```
$ gst-launch-1.0 --gst-debug=gesturedetect:INFO v4l2src ! jpegdec ! gesturedetect
     debug=1 debug-overlay=1 ! videoconvert ! ximagesink async=0 sync=0
```

During the developmental phase, we conducted experiments using various models featuring distinct input sizes. The models employed were Faster R-CNN, YOLO, and SSD, each exhibiting varying speeds, measured by the time taken to run inference on one image. The input size represents the image dimensions utilized for detection by the model, with larger images generally providing better accuracy, albeit with a corresponding increase in time taken to run inference. It is important to note that the models were ran on

Central Processing Units (CPUs) instead of Graphical Processing Units (GPUs), as Pexip runs everything on Central Processing Units (CPUs). Our aim was to achieve a detection rate of twice per second.

Performance was measured in FPS, and the data was gathered using a GStreamer plugin `fpsdisplaysink` [76]. Examples from it's documentation show how it can be utilized. By using the plugin, it logged the current and average FPS to the console. The exact pipeline utilized to test and read outputs is shown in code listing 3.3, which contains several GStreamer plugins, and the gesture detection element.

Code listing 3.3: gst-launch-1.0 pipeline utilized to collect FPS info.

```
$ gst-launch-1.0 v4l2src ! jpegdec ! gesturedetect ! videoconvert !
     fpsdisplaysink video-sink=ximagesink sync=0 -v 2>&1
```

In the full webapp integration, it was possible to dictate how often the element would receive video data. This interval is easily tweaked, along with the other properties of the element. This allowed rapid experimentation, such that it was possible to produce a better user experience in terms of how long it would take for a gesture to be detected and how strict the detector was.

3.5 Optimizations

Pexip requires optimization in terms of CPU and memory. To prevent unnecessary computation, the entirety of the detector preprocessing, referring to the image resizing, letterboxing, and layout reordering, is done in a single pass, instead of separating them into 3 methods. As the input images become larger, preproccesing may take too long, so it is important to use as few passes as possible.

3.6 Testing

3.6.1 Pexip codebase

Pexip heavily utilizes test-driven development, and because of this, a test suite for the detection element was created in the early stages, before any actual code was written. The test was slightly altered as the specifications for the element changed. Tests were written for the detection element itself, as well as it's integration in Mama, and a final test for it's integration with PMX. Every test utilized a specified set image, and tested whether the detector could detect a predefined gesture that was present on the image. The differences were only specific to the specific integration that was tested.

3.6.2 Model testing

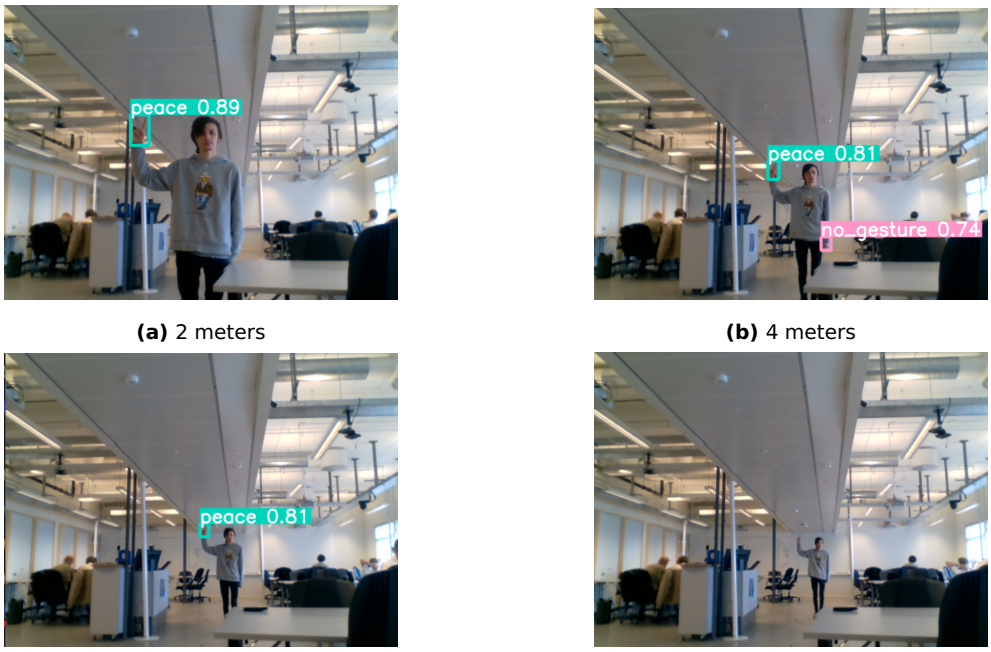
Testing of the different completed models were conducted, though in a smaller scale, due to time constraints.

Distance

A test was conducted for all YOLO models to try and see at what distances their results deteriorate. Deterioration in this test is defined as when the models can no longer:

- See any gestures.
- Classify gestures with confidence scores above 0.5.

The test would evaluate each model at increasing distances, starting at 2 meters and increasing by 2 meters for each step, up to 8 meters. A sample from the test is shown in Figure 3.5. Every test participant was asked to hold a "peace" sign in the air. Raw result values were gathered on a video, then averaged to produce the end result for each test.



(a) 2 meters

(b) 4 meters

(c) 6 meters

(d) 8 meters

Figure 3.5: A sample of the distance test. It shows the test subject holding a "peace" sign in the air. The images visualize detected gestures at their respective distances. For all but 8 meters, a "peace" gesture is detected.

User experience

Multiple smaller, casual tests were conducted to find the optimal duration a user would like to keep their hands raised to trigger the "raised hand" feature. This value had to be balanced between a good user experience, as well as being able to produce good detections to avoid false positives. These were conducted in several meeting rooms, and in home office settings with several users. The users would be within different distances to the camera, using different cameras, angles, and models. The results were individual preferences regarding what each test participant considered the optimal duration. Not

every result from this test was recorded. Instead, they were used for rapid experimentation, aiding in finding the optimal values for the meeting room environment in which it was conducted.

Chapter 4

Results

4.1 Model performance

The training of each model was conducted until either the mAP scores ceased to improve, or the total loss score exhibited minimal changes at a slow pace. Although these models could potentially yield better results with longer training durations, we made the decision to stop training considering the diminishing performance gains in relation to the additional training time required.

The primary evaluation metrics used were the mAP@50, mAP@[0.5:.95] scores, and inference times of the models.

4.1.1 Training results

As illustrated in Table 4.1, both YOLO models achieved notable mAP values. However, given their similar and high mAP@50 scores, we turned our attention to the mAP@[.5:.95] scores to see any substantial performance distinctions. The YOLOv8m model exhibited significantly higher accuracy at the higher IoU thresholds.

A significant disparity in loss values can be observed Table 4.2, where the YOLOv5 model produced a substantially lower total loss compared to YOLOv8.

In Table 4.1, the training results for the SSD and Faster R-CNN models are presented. It

Model	mAP@50	mAP@[.5:.95]	Recall
YOLOv5s	0.987	0.839	0.979
YOLOv8m	0.988	0.877	0.990
SSD	0.491	0.313	0.521
Faster-RCNN	0.964	0.726	0.778

Table 4.1: mAP@50,mAP@[.5:.95] scores, and recall for models

Model	Box Loss	Class loss
YOLOv5s	0.017	0.004
YOLOv8m	0.606	0.330

Table 4.2: Training loss for YOLO models

is evident that the SSD model achieved significantly lower values compared to the other models. The loss graphs in Figure 4.1 and Figure 4.2 for the two models outlines that the training was stopped when the significant changes in loss had stopped. We also see a artifact in the Faster R-CNN graph. This is due to that the training was halted, and then resumed.

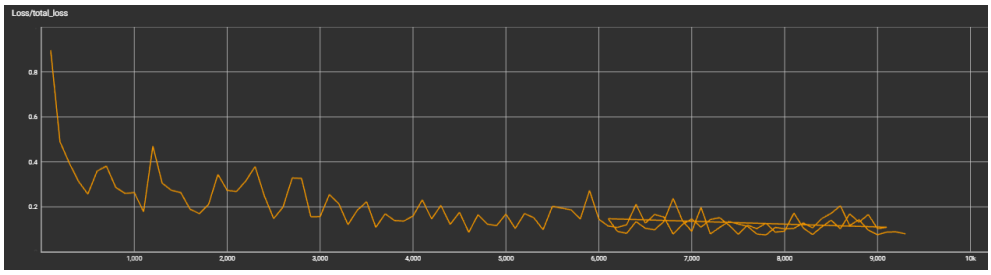


Figure 4.1: Graph of total loss versus steps for Faster R-CNN where loss is plotted against steps

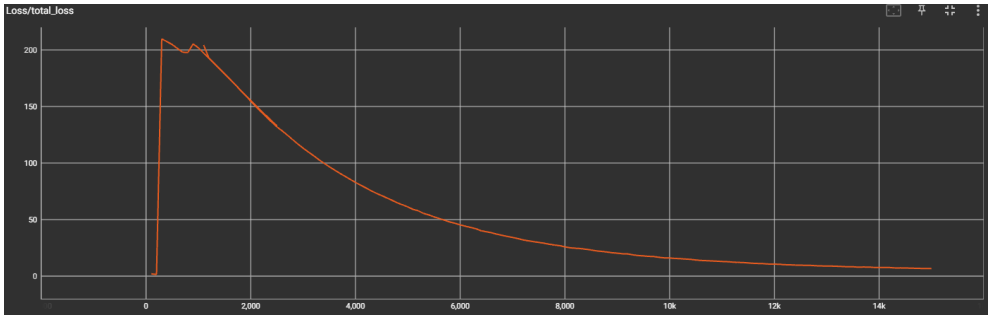


Figure 4.2: Graph of total loss versus steps for SSD where loss is plotted against steps

4.1.2 Distance test results

Referring to the distance test mentioned in Section 3.6.2, the results are shown in Table 4.3. The values in the distance columns are average values from 3 tests with 3 individuals. The results from the individual tests can be found in Appendix A. None of the models managed to detect anything at 8 meters, so this column was omitted. At 2 meters, all models performed similarly. At 4 meters, YOLOv5 and v8 performed similarly, while v7

Model	Avg. FPS	Input size	2m	4m	6m
YOLOv5	15.7	480 x 480	0.92	0.84	0.64
YOLOv7	26.3	320 x 320	0.88	0.59	>0.5
YOLOv8	3.3	640 x 640	0.88	0.81	0.61

Table 4.3: FPS performance of YOLO models running on a AMD Ryzen 7 PRO 4750U CPU in the GStreamer pipeline. Test results from a distance test is also shown, displaying the model confidence for the "peace" gesture at various distances.

had a significant drop in confidence. At 6 meters, v5 and v8 performed similarly, and v7's results had deteriorated to a degree that it could no longer detect any gesture with a confidence score of 0.5 or above.

4.2 GStreamer performance

As mentioned in 3.4, performance was measured in FPS. Both the video-stream and OpenVINO Runtime model inference were processed on an AMD Ryzen 7 PRO 4750U. GPUs were not available, and were not relevant as Pexip do not use GPUs. YOLOv5, YOLOv7, and YOLOv8 were tested with GStreamer and OpenVINO. Only YOLOv5 and YOLOv7 were fully implemented. The training of the YOLOv8 model was not completed in time to allow spending resources on it's full implementation. It was only partially implemented, allowing monitoring of it's framerate. YOLOv5 resulted in an average of 15.7 FPS, YOLOv7 displayed an average FPS of 26.3, and YOLOv8 displayed an average FPS of 3.3. The detailed results can be seen in Table 4.3.

4.3 Video call user experience

No real metric was found to measure the user experience in a video call. However, multiple informal tests were conducted as mentioned in Section 3.6.2.

It was found that most users were content with holding their hands still in the air for about 1.5 seconds, corresponding to a configuration of minimum 3 detections and 2 detections per second to trigger the raise hand function. Some of the test subjects thought this speed was too slow. However this was considered the optimal configuration to avoid false positives. The quality of the camera and the background in which one raised their hands were found to be important. Raising hands against plainer backgrounds with less noise, such as a white background and minimal sunlight pointing directly at the camera, performed better.

As users moved further away from the camera, the quality of the detections deteriorated, causing the "palm" and "stop" gestures to be mistaken for another if users did not make it very clear which gesture they wanted to exhibit. Unless the users clearly spread

their fingers, or held them firmly together, when at a further distance, the models could mistake the "palm" for a "stop" or vice versa.

4.4 Pexip video conference

As a product of everything previously mentioned, it was possible to launch an instance of the Pexip webapp, and join as participants. Then, using minimum 3 detections, with 2 detections per second, it was possible to raise a hand into the air, and trigger the "raised hand" functionality, without clicking any buttons. This was made clear by a banner popping up. We could then lower our hand using the "fist" gesture. Figure 4.3 shows a video call using one of Pexip's video conference webapps, where the "raised hand" banner is shown as a result of having triggered the "raise hand" functionality by utilizing the gesture detector.

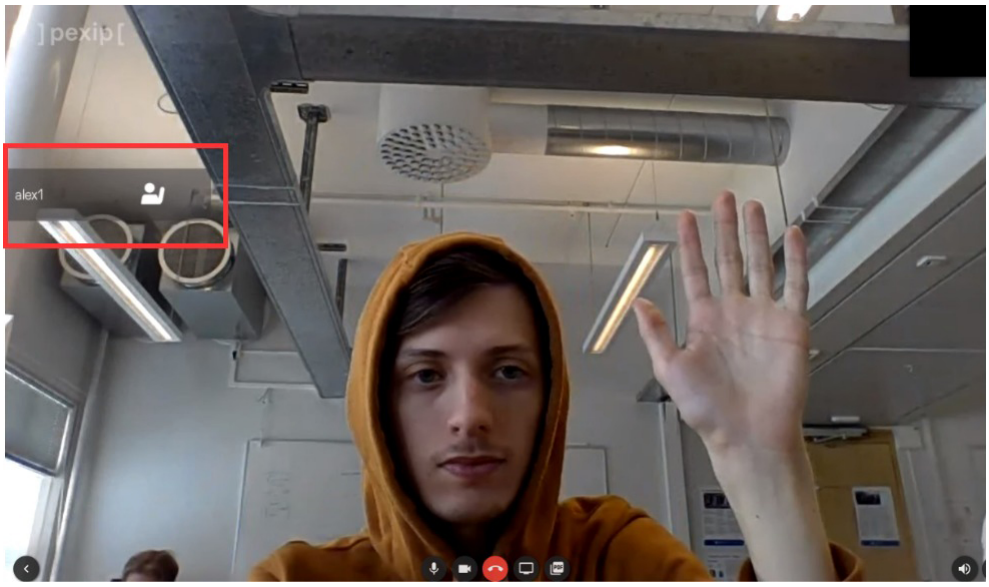


Figure 4.3: A Pexip call with the "raised hand" banner raised, shown within the red box, after triggering it with a "palm" gesture.

Chapter 5

Discussion

5.1 Interpretation of results

5.1.1 Trained models

Looking at the raw results from Section 4.1.1, it is clear that the YOLO models had the best metrics. Due to low mAP scores on the SSD's, it was not implemented in the GStreamer element. Faster R-CNN performed slightly worse at mAP@50, but had a significant drop within mAP@[.5:.95] and recall, compared to the YOLO model. This resulted in Faster R-CNN also not being implemented into the GStreamer element.

The reasoning for why SSD performed as badly as it did, is still unknown. The significantly lower mAP scores were unexpected. The mAP scores were almost 50% lower than the YOLO models. Generally, the results from SSD and Faster R-CNN can be expected to be within the same range. The reasoning for the bad scores was not investigated thoroughly because of time constraints.

The large loss disparity between YOLOv5 and v8, can be explained by a different loss calculation within the two versions.

5.1.2 Model usage

For model usage, SSD and Faster R-CNN is omitted, due to their lack of good comparative performance against YOLO models. Additionally, the pre-trained v7 model mentioned in 3.1.2 was used in the comparison, which allowed us to compare it against our own models.

The results show that there is a connection between the input image sizes and the model inference speed. The larger the input images are, the slower the algorithms will run. The average FPS for each model allowed us to consider whether each one *could* even be considered to be used in a real scenario. With v8 displaying an average FPS of only 3.3,

it is not sufficient.

The distance tests show a positive correlation between image input size, and the ability of the model to detect objects further away. YOLOv7 was unable to detect any gestures at a distance of 6 meters, while v5 and v8 were able to do so, with confidence scores above 0.5. It is indicative that this is due to v7's smaller input image size (320 x 320), compared to v5 and v8, which used larger input image sizes (480 x 480 and 640 x 640 respectively). Despite YOLOv8 using an input image size of 640 x 640, it performed worse than YOLOv5, despite consisting of newer technology. This was unexpected, as it should have been an *improvement* to v5. At every distance, YOLOv5 had the best performance, even though it's image input size was not the largest.

It is worth noticing the difference in performance scores of v8 and v5. The metrics of v8 were lower, and came with the drawback of significantly more time spent running inference, and produced a lower average FPS. In comparison, v5 produced better metrics with a 475% increase in FPS.

YOLOv5 is recommended to utilize of the models that were trained, because it had the best overall performance. YOLOv8 had a lack of performance, and YOLOv7 was unable to detect gestures from a 6 meter distance.

5.1.3 User experience

It is important to note that the results from the informal user experience tests, reflect the personal preferences of each individual. Therefore, these results should be interpreted cautiously, considering that user preferences may vary. The tests showed an optimal duration required to display a gesture, of 1.5 seconds. This duration may be suitable, as it shaves off a lot of possible false positives. It also means that a user must have a clear intention of raising their hands if they want to utilize the "raised hand" feature.

The observation that certain gestures may be confused by the detector, illustrates that a user may have to, in some cases, clearly delineate the intended gesture. An example would be the "palm" and "stop" gestures. The detector may sometimes confuse these two gestures, which can cause detections to be disposed due to different labels, despite the user expressing the same intended gesture. Therefore, it could be reasonable to combine these gestures into groups. These groups would enable the detection algorithm to allow all gestures within individual groups to be considered equal. Then, instead of comparing only the individual labels, it would compare these groups. This would help to provide a more seamless and less confusing user experience.

5.2 Alternative approach

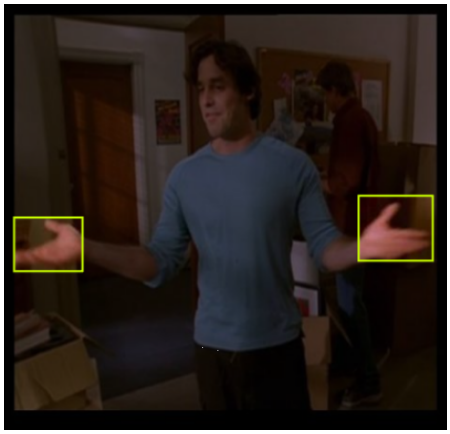
An alternative approach was also experimented with. This approach was proposed in the early stages of the project, before extensive research in different object detection architecture was done, and resembles the architecture of Faster R-CNN. The approach involved using a model to detect all the different gestures. The model would create a bounding box around the hands in a picture and then send the proposed regions to a classifier model.

To develop an effective hand gesture detection model, obtaining a suitable dataset was necessary. Datasets that could detect hands or hand gestures were searched for. Initially, several smaller datasets were found and tested using the YOLOv5s model.

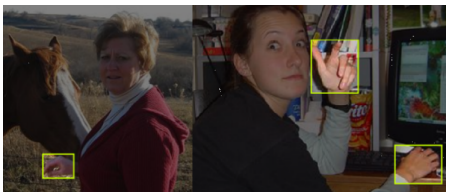
These datasets proved sufficient for individual cases, but they lacked quantity and diversity, rendering them unsuitable for a general solution.

After conducting further research, a dataset that aligned better with the project requirements was discovered. Arpit Mittal, Andrew Zisserman, and Philip H. S. Torr compiled this dataset by capturing snapshots from movies [77]. It featured a wide range of pictures and was sufficiently large for the project's needs. Permission to utilize this dataset was obtained from the creators, and approximately 1500 pictures were extracted from it.

To streamline the labeling process, the dataset underwent relabeling as it was found that converting the labeling from MATLAB [78] was more challenging than manually relabeling it. The relabeling task was accomplished using the internal labeling tool provided by Roboflow. To enhance the dataset's diversity, it was merged with two publicly available datasets [79, 80]. The initial models trained on the dataset showed poor performance when dealing with close-ups of hands. This led to the incorporation of additional datasets. In total, the dataset comprised around 2000 images and yielded promising results with limited training time using YOLOv5. One can see a snippet of this dataset with annotations in Figures 5.1a and 5.1b .



(a) Image from movie dataset



(b) Image from public dataset

Figure 5.1: Figure contains two pictures from our dataset. Both with bounding boxes on their hands.

5.2.1 Classifier

During the research process, a multitude of gesture classification datasets were discovered. However, the majority of these datasets were deemed unsuitable for the project due to various limitations. Certain datasets had insufficient data, while others focused on video-based action gestures, which did not align with the objective of static hand gesture classification. Furthermore, a few datasets were designed specifically for gesture recognition from a first-person perspective.

Later, we settled on a classifier dataset [81] to experiment with, even though it looked rather lacking in variety of the pictures. The dataset incorporated contrast masking of the hand gestures as depicted in Figure 5.3a.

5.2.2 Model training

The hand detection model employed YOLOv5, chosen for its comprehensive documentation and satisfactory mAP scores on the COCO dataset. The training process occurred on the Idun cluster [69]. The model underwent training and was subsequently evaluated using mAP scores. The training process utilized the "train.py" script from the Ultralytics repository [46], with adjustments made to parameters like batch size and image size. Satisfactory mAP scores were achieved, leading to the continuation of the planned development of the classifier. Additionally, real-time demonstrations were performed on local systems

The classifier was designed using a basic neural network that accepted a 40x40 black and white input, as required by the classifier's specific format. The classifier was trained using data augmentation to introduce noise, enhancing its generalizability and robust-

ness. A snippet of the dataset after augmentations can be seen in Figure 5.3b. The model underwent training for a few minutes until it achieved a high level of accuracy on the dataset as seen in Figure 5.2.

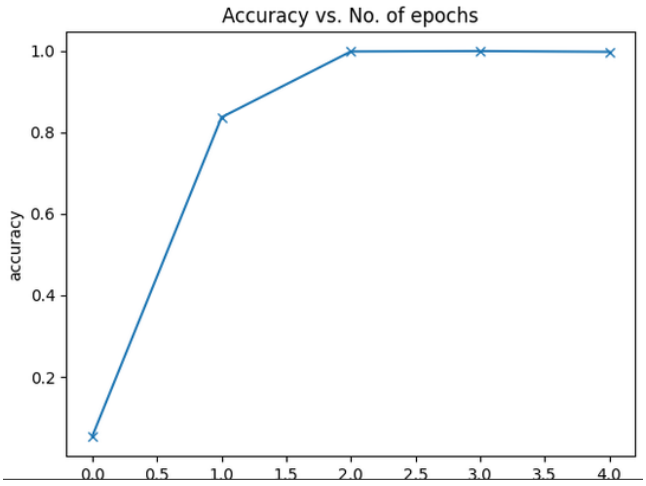


Figure 5.2: Accuracy of classifier [81].

(a) Image from the classifer dataset [81].

(b) Augmentation of classifier dataset [81].

Figure 5.3: Figure displaying the dataset snippet, and the same snippet after various augmentations.

5.2.3 Results

The bounding boxes generated by the hand detection model were extracted, and the hands were cropped and resized to the appropriate dimensions. These resized hands were then fed into the classifier, as seen in Figure 5.4a. Unfortunately, even after altering the training data with different transformations, the model was unable to detect the right gestures, as seen in Figure 5.4b. This might have been more successful if we had a larger

dataset or applied more transformations to it. However, upon further consideration, this approach could have larger overhead compared to the approach that was ultimately selected.

(a) Image of hand cropped from the hand detection model and resized to fit the classifier model.

```
Predictions: [6, 6, 6, 6]
Actuall:     [0, 3, 7, 9]
```

(b) classifier predictions versus actual values, where the numbers symbolize the type of gesture it predicts the image contains.

5.3 Implementation challenges

5.3.1 Idun / GPU cluster challenges

We conducted training on the Idun GPU cluster, which proved to be effective for the YOLO models. However, when it came to the TensorFlow models, we encountered various issues. Specifically, there were two problems that we struggled to address.

Firstly, we encountered segmentation errors during the startup of the training script. This type of error typically arises when a program attempts to write to read-only memory. It's possible that these errors were triggered by different user configurations or similar factors, but we could not pinpoint the exact cause.

Secondly, we encountered intermittent CORE DUMP errors at different intervals. These errors occurred consistently after running the models for a maximum of 1000 steps. As a result, we were unable to continue training the TensorFlow models on the Idun GPU cluster.

Consequently, we made the decision to train all further TensorFlow models using another service named Lambda Labs [82], which is a service that provides cloud GPU clusters. This alternative service helped us overcome the issues we encountered on the Idun cluster, enabling us to complete the training successfully.

5.3.2 GStreamer element

Moving from C to C++

It was found that using the OpenVINO C API was not ideal, as every attempt to load any model was unsuccessful. The documentation for the C API for this version of OpenVINO was not found to be helpful. In most cases, it only had relevant documentation for Python or C++. As there was no apparent solution to this issue, another attempt was made using C++ instead, which did not have the same issue. Using C++, OpenVINO successfully loaded a test model. Due to this, a switch and quick rewrite was made from C to C++ for all relevant files.

OpenVINO and ONNX versioning

Initially, we used the OpenVINO Model Optimizer to convert our `.onnx`-files to `.bin` and `.xml`-files. Through several experiments, we found that it was not possible to convert a number of `.onnx`-files, because the version of the Model Optimizer was not compatible with the operator set version of the files. We made attempts with several Python scripts, setting the versions manually and loading the models directly and then changing them. However, it did not work. We found later that it was possible to simply load `.onnx` files, *but only if* the ONNX operator set version were compatible with the OpenVINO Runtime. After training the models themselves, their implementations allowed us to specify the operator set that we wanted. After testing multiple operator sets, we found that operator set version 13 was compatible.

5.3.3 Time constraints

If the amount of model parameters, size of input images, and architecture complexity increases, so does the time it takes to train models using these architectures. We faced some time constraints due to the sheer time it took to train a number of the models due to this, and time spent on setting up individual training environments. As we desired to benchmark the performance of several models, we faced bottlenecks due to the time it took to upload datasets to the servers they were trained on, and the time it took to train them. There was more to this thesis than training models, so we could not spend *all* our time on it. This left us with a smaller gap to undergo the training. To counteract this, some shortcuts were taken. Only a subset of HaGRID was used. This caused less time spent waiting for the transfer of data, as well as training time. This allowed for training of multiple models within a decent timeframe. If more time had been available, the whole dataset would have been used. Preferably, it would be attempted to gather image data from specific environments, such as meeting rooms or offices, in addition using HaGRID.

Detectron2 [83] was also tested. Due to an issue with inconsistent labeling on the different models from the Detectron2 Model Zoo and time constraints, we were not able to train enough models with this to include it in our thesis.

5.3.4 Corrupt training data

When formatting HaGRID, one of the TFRecord files crashed the training. It took some time to troubleshoot because the reason behind it was an extra space character in the label names, this caused the correlated label annotations not to match up, and gave us an extra bit error when training the SSD and Faster R-CNN models. Determining the cause of the additional bit errors we encountered was a time-consuming task since there were no clear indicators suggesting that this error was the underlying reason.

5.3.5 Usage in commercial setting

Early on, we were trying to make sure everything we used was accepted to use in a commercial setting. As the due date came closer, we figured out that we did not have time to ensure all datasets and architectures had licenses that allowed for free use in a commercial setting. If Pexip wants to use this feature in a commercial setting, it is recommended to inspect the relevant licenses.

5.3.6 Poor documentation

A large amount of the repositories with implementations of object detection models had poor documentation. The recurring problems were often that there was little to no documentation on how to train with a custom data set, despite parts of their documentation specifying that this was possible. A lot of time was spent trying to make sense of this, and a lot of trial and error went into making them work.

Some major issues were also that some implementations needed a different annotation format than the one we had available. Attempts were made to try and find any reasonable converters, but this proved to be difficult. We did not find any converters that worked adequately. This issue was especially prominent with Single Shot MultiBox Detector (SSD) implementations. However, with TensorFlow Object Detection API this was not an issue due to them using TFRecords for all model training.

5.3.7 Uncertainty of requirements

Due to uncertainties regarding the requirements on our part, we lacked clarity on whether to prioritize performance (framerate) or accuracy (mAP). Furthermore, we did not receive sufficient information regarding the intended use case, resulting in a potential optimization for different use cases than the intended one. Our hypothesis was that ensuring functionality under poor conditions, such as participants with inadequate lighting or cluttered backgrounds, was crucial. During our visit to their office, we recognized that one of the desirable conditions for our gesture detector was a meeting room setting. In meeting rooms, we observed a notable issue wherein the detector’s performance was unsatisfactory when participants were not directly facing the camera. This issue could potentially be mitigated through the utilization of data augmentations, such as perspective transformation, during the training process.

5.4 Limitations

5.4.1 Dataset classes

HaGRID was embodied as the final dataset in which all our models were trained on. However, it is important to note that, while the gestures it contains are plentiful, there are a magnitude of possible gestures that one could desire to have access to, which are not available. An example of this could be sign language. It is not available in HaGRID, so other datasets would have to be explored. Regardless, due to the context, the sheer size of the dataset, and lack of other similar datasets, HaGRID was regarded as the best choice.

5.4.2 Assumptions and constraints

OpenVINO version and OpenCV In early stages, we were notified that usage of OpenCV [84] was not allowed. This was a major constraint, because of the vast image processing capabilities OpenCV provides. It made it impossible to use several preprocessing capabilities of OpenVINO, which required OpenCV. However, the development was manageable without OpenCV.

As we were informed that we could not use OpenCV, we made the assumption that we could not change or update other software within the Pexip development Docker container. One such software was OpenVINO. As of now, the OpenVINO version utilized is 2021.4, while the latest is 2022.4. The latest version contains a multitude of quality-of-life features with a new API version, with better documentation.

Chapter 6

Conclusion and further work

In conclusion, this thesis has explored the creation and implementation of a "raised hand" detector which can be utilized within a Pexip conference call. Widely known object detection architectures were used and trained with the HaGRID dataset, specializing them for gesture recognition. These highly performant models were then employed to enhance user interactivity during Pexip video conference calls. With optimization in mind, it provides the full capability to detect a user raising their hand, and have it trigger the "raised hand" functionality, without consuming vast amounts of CPU or memory resources.

Through a comprehensive analysis of different object detection architectures and their implementations, it has been established that the YOLO models provide the best overall performance. The most useful model was found to be YOLOv5, with an input image size of 480x480. By utilizing YOLOv5 and raising a gesture for 1.5 seconds, it allowed for seamless interactivity within a video call, relieving the users of the need to touch their computers if they want to raise their hands.

Multiple approaches were attempted and discussed to find the optimal solution to detecting hand gestures. The stepwise approach in which hands are detected and then put through a classifier, and simply considering gestures as different objects and employing state-of-the-art object detection algorithms. The latter was found to be the better, and was expanded further upon and finally completely implemented into the Pexip codebase.

While we consider our solution as a strong foundation for a module which Pexip can utilize, it is important to remember that there is room for expanding the set of available gestures, if more gesture datasets appear, or are created.

In terms of future work, it is recommended to further research several aspects of the current solution, such that it can be considered a commercially viable product. The use-cases can be increased, the detection algorithm can be expanded upon, and tinkering

with larger model input sizes and parameters can improve the detector. These ideas are explored more in Section 6.1.

Overall, the successful implementation of the gesture detection module into Pexip's video conference platform offers promising opportunities for improved user interactivity. By refining the detector and conducting more formal usability tests, it can provide more immersive and effective means of communication between the conference participants.

6.1 Future work

6.1.1 Future models

Due to the active field of object detection, technological advancements in the form of new architectures will keep surfacing. As newer architectures appear, they have the potential to possess enhanced complexity, which may produce better performance compared to old architectures. By keeping up with these advancements, it is possible to re-train the gesture detection model with these new architectures. If newer models are found to be better, it ensures a high quality product. As an example, near the end of writing, a new YOLO architecture (YOLO-NAS) appeared, which, according to Deci.ai, outperforms previous architectures [85].

6.1.2 Gesture detection usecases

As of this thesis, the gesture detection's only purpose is to trigger the raised hand functionality. However, as there are multiple other gestures available, and due to the fact that humans may raise their hands for multiple reasons, some additional future use cases could be considered. Some features that are suggested are muting yourself with the "mute" gesture, and incorporating a voting feature with the thumbs up (like) and thumbs down (dislike) gestures.

Counting raised hands

If a user wants to gather a quick response about e.g. they agree to something, they may create an event in which detector counts the raised hands for participants. E.g. "raise your hand if you agree we should do this", followed by users raising their hands, then the server can compute the amount of raised hands, and display it. This can also be combined with a face detector to make sure that someone is not cheating by raising two hands, despite being only one person on the video.

Polls

The data set contains both a thumbs up and a thumbs down. With these gestures implemented, it is possible to create simple polls, where the users can simply raise a thumbs up or thumbs down to respond to the poll in real-time.

Add visuals for raised hand to video

If there are participants from several video-conference platforms that are in a call, those platforms may not have given Pexip the ability to interoperate the raised hand functionality between their platforms. One way to work around this is to add video visuals that a user has raised their hands, such as an icon on their video.

6.1.3 Detection algorithm

The detection algorithm can be easily extended and changed. As it's purpose is to detect high quality gestures and reduce false positives, it may become too strict for some cases. Thus, some changes can be considered.

Combining certain gestures

The algorithm strictly requries that gesture labels are the same. However, if a gesture is further away, it may be hard to distinguish some of them, causing the labels to jump back and forth. By mapping some gestures together in groups, the user experience may improve. One such example are the `palm` and `stop` gestures. Allowing the algorithm to consider these as the same gesture could cause detections to be smoother.

Supermajority voting

Instead of requiring every individual detection before a signal can be emitted to be the same, a voting system can be considered. If 10 consecutive detections are required, it will as of now require all 10 to be contain the same label. By changing this such that it requires a fraction instead, it may improve the user experience. E.g. the 10 consecutive detections require *at least* 8 out of the 10 detections to be of the same label. This may prevent some false negatives if the detector is confusing two similar labels.

6.1.4 Other

More usability tests

While we found a set of optimal values for the duration in which a user must keep their hand raised, we only conducted casual tests. It would be very beneficial to perform more formal usability tests to ensure the best configuration for the userbase of the product.

Input image size

The underlying models may be changed to improve the accuracy. We did not have the resources, both time and money, to be able to train a large-scale model. By training the models on higher resolutions, they have a better ability to detect smaller objects. Therefore, a possible improvement is to train a model with a larger input size to improve the accuracy of gestures from further away. Balancing the desired accuracy of a model and the CPU resources needed for a more complex model is required. As shown in results,

running a model at 640 x 640 compared to 320 x 320 will show a significant increase in inference time.

Model parameter size

The detection models may be trained on networks with more model parameters. Increasing the model size will cause it to be slower, which means some balancing will be required.

6.1.5 Input image splitting

A letterboxing effect is used on the input data to the detection models, which may cause large parts of the input images to become black bars to help maintain the aspect ratio. A possible alternative could be splitting the input images into smaller images and pass those through the model instead of the full letterboxed and resized image. This could prevent some of the loss of image quality, and maybe improve the overall model metric performance. However, it would likely increase the inference time.

6.1.6 Combine detector with a motion detector

A motion detector is simply a module which detects motion. This could be useful to detect only the moving parts of a video, such as the motion of raising a hand. The motion detector would pass the movement region to the gesture detector, such that it only runs inference on the smaller region. This could decrease the size of images passed to the gesture detector, making it less expensive to use. However, it could be challenging to detect consecutive detections, as the motion detector would only detect the initial movement. A possible solution could be to track the moving regions, then run future detections on areas based on the previous regions, $\pm$ some margin of error.

Broader impact

Environment For resource-intensive applications, such as real-time video transcoding, the workload may contribute to increased energy consumption and costs. It should be a primary objective to ensure that energy consumed comes from a clean source, and to use energy-efficient components to lessen the energy needed to do the same computations.

GPU clusters As the field of AI has recently experienced rapid development, especially with large language models and data processing, it has been found that training some AI models can create nearly five times the lifetime emissions of the average American car [86]. These models are often trained on large GPU clusters. These clusters consume massive amounts of electricity and create additional electronic waste. It is important for engineers to create sustainable systems, and for those who do train said models to be responsible for possible carbon emissions. Due to the extreme capabilities of GPUs within AI, researching the services beforehand is advised to be aware of how their services consume power.

Privacy concerns The use of object detection has the potential to cause privacy concerns, due to its possible use within surveillance. Utilizing such algorithms in certain spaces can raise concerns about collection and use of personal user data, without the user explicitly complying.

Automation and job displacement With more automation, comes the ability to replace human workers to do labor in various industries. While automation can improve efficiency and productivity, it may also cause human workers to become unemployed, which then can cause socio-economic challenges.

Accessibility Object detection can be employed to create more accessible environments, such as assistive technology which can be used to detect gestures or sign language for mute individuals.

Bibliography

[1] 2023. [Online]. Available: https://www.mongodb.com/unstructured-data/.

[2] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org.

[3] T. Mitchell, *Machine Learning* (McGraw-Hill series in computer science), en. New York, NY: McGraw-Hill Professional, Mar. 1997.

[4] D. Ciresan, A. Giusti, L. Gambardella, and J. Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/459a4ddcb586f24efd9395aa7662bc7c-Paper.pdf.

[5] D. Ciresan, U. Meier, J. Masci, and J. Schmidhuber, "A committee of neural networks for traffic sign classification," *Proceedings of the International Joint Conference on Neural Networks*, pp. 1918–1921, Jul. 2011. DOI: 10.1109/IJCNN.2011.6033458.

[6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL].

[7] S. Levine, C. Finn, T. Darrell, and P. Abbeel, *End-to-end training of deep visuomotor policies*, 2016. arXiv: 1504.00702 [cs.LG].

[8] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, 2012.

[9] F. Sebastiani, "Machine learning in automated text categorization," *ACM Computing Surveys*, vol. 34, no. 1, pp. 1–47, Mar. 2002. DOI: 10.1145/505282.505283. [Online]. Available: https://doi.org/10.1145/505282.505283.

[10] Kowsari, J. Meimandi, Heidarysafa, Mendu, Barnes, and Brown, "Text classification algorithms: A survey," *Information*, vol. 10, no. 4, p. 150, Apr. 2019, ISSN: 2078-2489. DOI: 10.3390/info10040150. [Online]. Available: http://dx.doi.org/10.3390/info10040150.

[11] M. Din, R. Ratan, A. K. Bhateja, and A. Bhateja, “Multimedia classification using ann approach,” in *Proceedings of the Second International Conference on Soft Computing for Problem Solving (SocProS 2012), December 28-30, 2012*, B. V. Babu, A. Nagar, K. Deep, M. Pant, J. C. Bansal, K. Ray, and U. Gupta, Eds., New Delhi: Springer India, 2014, pp. 905–910, ISBN: 978-81-322-1602-5.

[12] *IBM deep learning*, https://www.ibm.com/topics/deep-learning, Accessed: May 7, 2023, 2023.

[13] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. DOI: 10.1038/nature14539.

[14] *Midjourney*, 2022. [Online]. Available: https://www.midjourney.com/home/.

[15] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, *Zero-shot text-to-image generation*, 2021. arXiv: 2102.12092 [cs.CV].

[16] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, *High-resolution image synthesis with latent diffusion models*, 2022. arXiv: 2112.10752 [cs.CV].

[17] *Introducing chatgpt*, 2022. [Online]. Available: https://openai.com/blog/chatgpt.

[18] D. Shiffman, “The nature of code: Simulating natural systems with processing,” in self-published, 2012, ch. 10: Neural Networks. [Online]. Available: https://natureofcode.com/book/chapter-10-neural-networks/.

[19] J. B. Ahire. “The artificial neural networks handbook (part 1).” (2018), [Online]. Available: https://www.datasciencecentral.com/the-artificial-neural-networks-handbook-part-1/.

[20] M. A. Nielsen, *Neural Networks and Deep Learning*, en. Determination Press, 2015.

[21] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[22] 2023. [Online]. Available: https://citeseerx.ist.psu.edu/doc/10.1.1.17.1332.

[23] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *J. Mach. Learn. Res.*, vol. 12, no. null, pp. 2121–2159, Jul. 2011, ISSN: 1532-4435.

[24] L. Prechelt, “Early stopping — but when?” In *Neural Networks: Tricks of the Trade: Second Edition*, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 53–67, ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_5. [Online]. Available: https://doi.org/10.1007/978-3-642-35289-8_5.

[25] N. Shahriar. “What is convolutional neural network (cnn) - deep learning.” (2023), [Online]. Available: https://nafizshahriar.medium.com/what-is-convolutional-neural-network-cnn-deep-learning-b3921bdd82d5.

[26] A. Kumar. “Real-world applications of convolutional neural networks.” Accessed: May 18, 2023. (2021), [Online]. Available: https://vitalflux.com/real-world-applications-of-convolutional-neural-networks/.

[27] *Convolutional neural networks*, 2023. [Online]. Available: https://buffml.com/convolutional-neural-networks/.

[28] *Max pooling*, 2023. [Online]. Available: https://paperswithcode.com/method/max-pooling.

[29] K. Fukushima, "Visual feature extraction by a multilayered network of analog threshold elements," *IEEE Transactions on Systems Science and Cybernetics*, vol. 5, no. 4, pp. 322–333, 1969. DOI: 10.1109/TSSC.1969.300225.

[30] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, 2019. DOI: 10.1186/s40537-019-0197-0.

[31] T. G. Karimpanal and R. Bouffanais, "Self-organizing maps for storage and transfer of knowledge in reinforcement learning," *Adaptive Behavior*, vol. 27, no. 2, pp. 111–126, Dec. 2018. DOI: 10.1177/1059712318818568. [Online]. Available: https://doi.org/10.1177%5C%2F1059712318818568.

[32] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV].

[33] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *Imagenet large scale visual recognition challenge*, 2015. arXiv: 1409.0575 [cs.CV].

[34] R. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, 2014. arXiv: 1311.2524 [cs.CV].

[35] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection*, 2016. arXiv: 1506.02640 [cs.CV].

[36] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," in *Computer Vision – ECCV 2016*, Springer International Publishing, 2016, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2. [Online]. Available: https://doi.org/10.1007%5C%2F978-3-319-46448-0%5C_2.

[37] P. Jaccard, "The distribution of the flora in the alpine zone.1," *New Phytologist*, vol. 11, no. 2, pp. 37–50, 1912. DOI: https://doi.org/10.1111/j.1469-8137.1912.tb05611.x. eprint: https://nph.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-8137.1912.tb05611.x. [Online]. Available: https://nph.onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-8137.1912.tb05611.x.

[38] R. Restrepo, *Intersect over union*. [Online]. Available: http://ronny.rest/tutorials/module/localization%5C_001/iou/.

[39] A. Anwar. "What is average precision in object detection localization algorithms and how to calculate it." (2022), [Online]. Available: https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b.

[40] T. Y. Lin, S. Pepose, R. Girshick, and Y. Wu, *Cocoeval.py*, GitHub, https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocotools/cocoeval.py#L501, 2019.

[41] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, *Microsoft coco: Common objects in context*, 2015. arXiv: 1405.0312 [cs.CV].

[42] X. Zhu, S. Lyu, X. Wang, and Q. Zhao, *Tph-yolov5: Improved yolov5 based on transformer prediction head for object detection on drone-captured scenarios*, 2021. arXiv: 2108.11539 [cs.CV].

[43] C.-A. Brust, T. Burghardt, M. Groenenberg, C. Kading, H. S. Kühl, M. L. Manguette, and J. Denzler, "Towards automated visual monitoring of individual gorillas in the wild," in *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2017, pp. 2820–2830. DOI: 10.1109/ICCVW.2017.333.

[44] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010. DOI: 10.1109/TPAMI.2009.167.

[45] J. Terven and D.-M. Cordova-Esparza, *A comprehensive review of yolo: From yolov1 to yolov8 and beyond*, Apr. 2023.

[46] G. Jocher, *Yolov5 by ultralytics*, version 7.0, May 2020. DOI: 10.5281/zenodo.3908559. [Online]. Available: https://github.com/ultralytics/yolov5.

[47] ultralytics, *Ultralytics*, May 2023. [Online]. Available: https://github.com/ultralytics/ultralytics.

[48] R. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, 2014. arXiv: 1311.2524 [cs.CV].

[49] R. Girshick, *Fast r-cnn*, 2015. arXiv: 1504.08083 [cs.CV].

[50] S. Ren, K. He, R. Girshick, and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, 2016. arXiv: 1506.01497 [cs.CV].

[51] A. Kapitanov, A. Makhlyarchuk, and K. Kvanchiani, "Hagrid - hand gesture recognition image dataset," *arXiv preprint arXiv:2206.08219*, 2022.

[52] *What is the yolov8 pytorch txt annotation format?* [Online]. Available: https://roboflow.com/formats/yolov8-pytorch-txt.

[53] R. Khandelwal, *Coco data format for object detection*, Dec. 2019. [Online]. Available: https://towardsdatascience.com/coco-data-format-for-object-detection-a4c5eaf518c5.

[54] [Online]. Available: https://www.tensorflow.org/tutorials/load%5C_data/tfrecord.

[55] Roboflow, *Roboflow: A computer vision platform for training and deploying models*, https://roboflow.com, Accessed: May 15, 2023, 2021.

[56] *Git*, 2005. [Online]. Available: https://git-scm.com/.

[57] *Github*, 2023. [Online]. Available: https://github.com/.

[58] *Docker*, 2023. [Online]. Available: https://www.docker.com/.

[59] B. Kernighan and D. Ritchie, *The C programming language*, 2nd ed. Pearson, 1988.

[60] B. Stroustrup, *The C++ programming language*, 4th ed. Addison-Wesley Professional, 2013.

[61] *Gstreamer*, 2023. [Online]. Available: https://gstreamer.freedesktop.org/.

[62] *Gstreamer bins*, 2023. [Online]. Available: https://gstreamer.freedesktop.org/documentation/application-development/basics/bins.html?gi-language=c.

[63] *Openvino*, 2023. [Online]. Available: https://docs.openvino.ai/latest/home.html.

[64] Apr. 2023. [Online]. Available: https://pypi.org/project/pip/.

[65] *Python*, 2023. [Online]. Available: https://www.python.org/.

[66] *Pytorch*, 2023. [Online]. Available: https://pytorch.org/.

[67] *Tensorflow*, 2023. [Online]. Available: https://www.tensorflow.org/.

[68] *Onnx*, 2023. [Online]. Available: https://onnx.ai/.

[69] *Ntnu idun*, 2022. [Online]. Available: https://www.hpc.ntnu.no/.

[70] 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, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, 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*, Nov. 2015. DOI: 10.5281/zenodo.4724125.

[71] C. Lee, *Zzanzu/yolo-convert-txt-2-xml: Darknet txt to darkflow xml*. [Online]. Available: https://github.com/ZZANZU/YOLO-convert-txt-2-xml.

[72] D. Tran, *Raccoon_dataset generate_tfrecord.py*, 2018. [Online]. Available: https://github.com/datitran/raccoon_dataset/blob/master/generate_tfrecord.py.

[73] *Ultralytics*, May 2023. [Online]. Available: https://pypi.org/project/ultralytics/.

[74] TensorFlow, *Tensorflow 2 detection model zoo*, 2016. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md.

[75] *Gstreamer properties*, 2023. [Online]. Available: https://gstreamer.freedesktop.org/documentation/plugin-development/basics/args.html.

[76] *Fpsdisplaysink*, 2023. [Online]. Available: https://gstreamer.freedesktop.org/documentation/debugutilsbad/fpsdisplaysink.html.

[77] A. Mittal, A. Zisserman, and P. H. S. Torr, "Hand dataset," [Online]. Available: https://www.robots.ox.ac.uk/~vgg/data/hands/.

[78] 2023. [Online]. Available: https://www.mathworks.com/products/matlab.html.

[79] jenny122419gmail.com, *Paperhandpose2.0 dataset*, https://universe.roboflow.com/jenny122419-gmail-com/paperhandpose2.0, Open Source Dataset, visited on 2023-02-10, Apr. 2022. [Online]. Available: https://universe.roboflow.com/jenny122419-gmail-com/paperhandpose2.0.

[80] opensource, *Total_hand dataset*, https://universe.roboflow.com/opensource-zzhlk/total_hand, Open Source Dataset, visited on 2023-02-10, Nov. 2022. [Online]. Available: https://universe.roboflow.com/opensource-zzhlk/total%5C_hand.

[81] *Hand gesture classifier dataset*, https://www.kaggle.com/datasets/aryarishabh/hand-gesture-recognition-dataset, Accessed: 2023-02-10.

[82] *Lambda labs*, 2023. [Online]. Available: https://lambdalabs.com/.

[83] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, *Detectron2*, https://github.com/facebookresearch/detectron2, 2019.

[84] *Opencv*, 2023. [Online]. Available: https://opencv.org/.

[85] Deci.ai, *Yolo-nas*, Apr. 2023. [Online]. Available: https://github.com/Deci-AI/super-gradients/blob/master/YOLONAS.md.

[86] E. Strubell, A. Ganesh, and A. McCallum, *Energy and policy considerations for deep learning in nlp*, 2019. arXiv: 1906.02243 [cs.CL].

Appendix A

Result CSV tables

Sample/Distance	2m	4m	6m
Test 1.	0.94	0.86	0.55
Test 2.	0.89	0.81	0.81
Test 3.	0.94	0.86	0.55
Average	0.92	0.84	0.64

Table A.1: YOLOv5 distance test results

Sample/Distance	2m	4m	6m
Test 1.	0.82	0.64	>0.5
Test 2.	0.87	0.61	>0.5
Test 3.	0.89	0.52	>0.5
Average	0.88	0.59	>0.5

Table A.2: YOLOv7Tiny distance test results

Sample/Distance	2m	4m	6m
Test 1.	0.89	0.82	0.55
Test 2.	0.85	0.78	0.74
Test 3.	0.89	0.82	0.55
Average	0.88	0.81	0.61

Table A.3: YOLOv8m distance test results

Appendix B

Model training configuration files

B.1 SSD

```
model {
  ssd {
    num_classes: 19
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
    feature_extractor {
      type: "ssd_resnet50_v1_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 0.00039999998989515007
          }
        }
        initializer {
          truncated_normal_initializer {
            mean: 0.0
            stddev: 0.029999999329447746
          }
        }
```

```
    activation: RELU_6
    batch_norm {
      decay: 0.996999979019165
      scale: true
      epsilon: 0.0010000000474974513
    }
  }
  override_base_feature_extractor_hyperparams: true
  fpn {
    min_level: 3
    max_level: 7
  }
}
box_coder {
  faster_rcnn_box_coder {
    y_scale: 10.0
    x_scale: 10.0
    height_scale: 5.0
    width_scale: 5.0
  }
}
matcher {
  argmax_matcher {
    matched_threshold: 0.5
    unmatched_threshold: 0.5
    ignore_thresholds: false
    negatives_lower_than_unmatched: true
    force_match_for_each_row: true
    use_matmul_gather: true
  }
}
similarity_calculator {
  iou_similarity {
  }
}
box_predictor {
  weight_shared_convolutional_box_predictor {
    conv_hyperparams {
      regularizer {
        l2_regularizer {
          weight: 0.00039999998989515007
        }
      }
```

```
      initializer {
        random_normal_initializer {
          mean: 0.0
          stddev: 0.009999999776482582
        }
      }
      activation: RELU_6
      batch_norm {
        decay: 0.996999979019165
        scale: true
        epsilon: 0.0010000000474974513
      }
    }
    depth: 256
    num_layers_before_predictor: 4
    kernel_size: 3
    class_prediction_bias_init: -4.599999904632568
  }
}
anchor_generator {
  multiscale_anchor_generator {
    min_level: 3
    max_level: 7
    anchor_scale: 4.0
    aspect_ratios: 1.0
    aspect_ratios: 2.0
    aspect_ratios: 0.5
    scales_per_octave: 2
  }
}
post_processing {
  batch_non_max_suppression {
    score_threshold: 9.99999993922529e-09
    iou_threshold: 0.6000000238418579
    max_detections_per_class: 100
    max_total_detections: 100
    use_static_shapes: false
  }
  score_converter: SIGMOID
}
normalize_loss_by_num_matches: true
loss {
  localization_loss {
```

```
        weighted_smooth_l1 {
        }
      }
      classification_loss {
        weighted_sigmoid_focal {
          gamma: 2.0
          alpha: 0.25
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
train_config {
  batch_size: 16
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.03999999910593033
          total_steps: 25000
          warmup_learning_rate: 0.013333000242710114
          warmup_steps: 2000
```

```
        }
      }
      momentum_optimizer_value: 0.8999999761581421
    }
    use_moving_average: false
  }
  fine_tune_checkpoint: "pre-trained-models/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/checkpoint/ck
  num_steps: 15000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  use_bfloat16: false
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  label_map_path: "annotations/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "annotations/train.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "annotations/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "annotations/test.record"
  }
}
```

B.2 Faster R-CNN

```
model {
faster_rcnn {
num_classes: 19
image_resizer {
keep_aspect_ratio_resizer {
min_dimension: 640
```

```
max_dimension: 640
pad_to_max_dimension: true
}
}
feature_extractor {
type: 'faster_rcnn_resnet50_keras'
batch_norm_trainable: true
}
first_stage_anchor_generator {
grid_anchor_generator {
scales: [0.25, 0.5, 1.0, 2.0]
aspect_ratios: [0.5, 1.0, 2.0]
height_stride: 16
width_stride: 16
}
}
first_stage_box_predictor_conv_hyperparams {
op: CONV
regularizer {
l2_regularizer {
weight: 0.0
}
}
initializer {
truncated_normal_initializer {
stddev: 0.01
}
}
}
first_stage_nms_score_threshold: 0.0
first_stage_nms_iou_threshold: 0.7
first_stage_max_proposals: 300
first_stage_localization_loss_weight: 2.0
first_stage_objectness_loss_weight: 1.0
initial_crop_size: 14
maxpool_kernel_size: 2
maxpool_stride: 2
second_stage_box_predictor {
mask_rcnn_box_predictor {
use_dropout: false
dropout_keep_probability: 1.0
fc_hyperparams {
op: FC
```

```
regularizer {
l2_regularizer {
weight: 0.0
}
}
initializer {
variance_scaling_initializer {
factor: 1.0
uniform: true
mode: FAN_AVG
}
}
}
share_box_across_classes: true
}
}
second_stage_post_processing {
batch_non_max_suppression {
score_threshold: 0.0
iou_threshold: 0.6
max_detections_per_class: 100
max_total_detections: 300
}
score_converter: SOFTMAX
}
second_stage_localization_loss_weight: 2.0
second_stage_classification_loss_weight: 1.0
use_static_shapes: true
use_matmul_crop_and_resize: true
clip_anchors_to_image: true
use_static_balanced_label_sampler: true
use_matmul_gather_in_matcher: true
}
}

train_config: {
batch_size: 64
sync_replicas: true
startup_delay_steps: 0
replicas_to_aggregate: 8
num_steps: 25000
optimizer {
momentum_optimizer: {
```

```
learning_rate: {
cosine_decay_learning_rate {
learning_rate_base: .04
total_steps: 15000
warmup_learning_rate: .013333
warmup_steps: 2000
}
}
momentum_optimizer_value: 0.9
}
use_moving_average: false
}
fine_tune_checkpoint_version: V2
fine_tune_checkpoint: "pre-trained-models/faster_rcnn_resnet50_keras/checkpoint/ckpt-0"
fine_tune_checkpoint_type: "detection"
data_augmentation_options {
random_horizontal_flip {
}
}

max_number_of_boxes: 100
unpad_groundtruth_tensors: false
use_bfloat16: true  # works only on TPUs
}

train_input_reader: {
label_map_path: "annotations/train.record"
tf_record_input_reader {
input_path: "annotations/train.record"
}
}

eval_config: {
metrics_set: "coco_detection_metrics"
use_moving_averages: false
batch_size: 1;
}

eval_input_reader: {
label_map_path: "annotations/train.record"
shuffle: false
num_epochs: 1
tf_record_input_reader {
```

```
input_path: "annotations/test.record
}
}
```

B.3 YOLOv5s

```
weights: runs/train/exp26/weights/last.pt
cfg: ''
data: datasets/hagrid/data.yaml
hyp:
  lr0: 0.01
  lrf: 0.01
  momentum: 0.937
  weight_decay: 0.0005
  warmup_epochs: 3.0
  warmup_momentum: 0.8
  warmup_bias_lr: 0.1
  box: 0.05
  cls: 0.5
  cls_pw: 1.0
  obj: 1.0
  obj_pw: 1.0
  iou_t: 0.2
  anchor_t: 4.0
  fl_gamma: 0.0
  hsv_h: 0.015
  hsv_s: 0.7
  hsv_v: 0.4
  degrees: 0.0
  translate: 0.1
  scale: 0.5
  shear: 0.0
  perspective: 0.0
  flipud: 0.0
  fliplr: 0.5
  mosaic: 1.0
  mixup: 0.0
  copy_paste: 0.0
epochs: 200
batch_size: 30
imgsz: 480
rect: false
resume: true
```

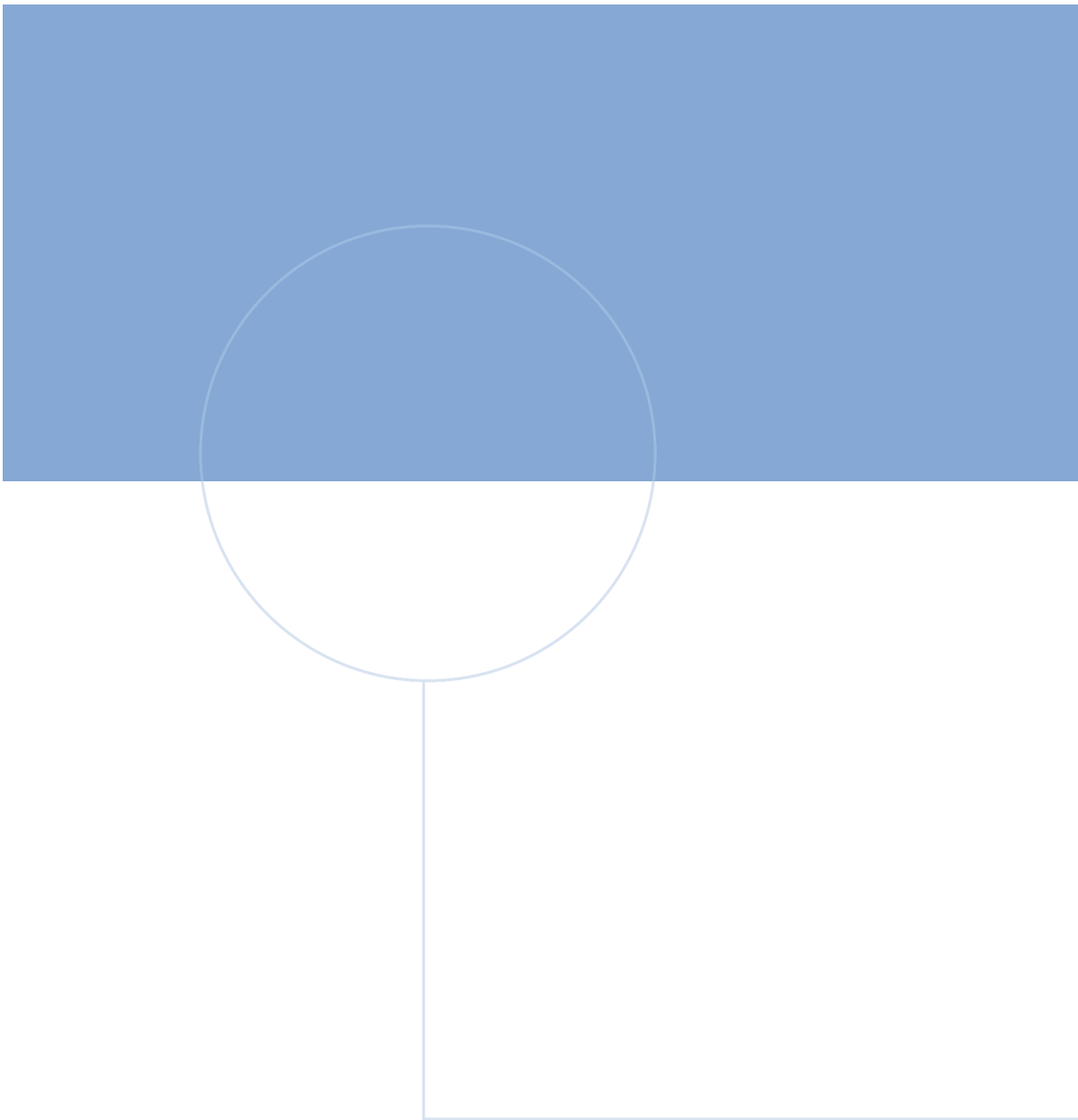
```
nosave: false
noval: false
noautoanchor: false
noplots: false
evolve: null
bucket: ''
cache: null
image_weights: false
device: 0,1
multi_scale: false
single_cls: false
optimizer: SGD
sync_bn: false
workers: 8
project: runs/train
name: exp
exist_ok: false
quad: false
cos_lr: false
label_smoothing: 0.0
patience: 100
freeze:
- 0
save_period: 10
seed: 0
local_rank: -1
entity: null
upload_dataset: false
bbox_interval: -1
artifact_alias: latest
save_dir: runs/train/exp26
```

B.4 YOLOv8m

```
task: detect
mode: train
model: yolov8m.pt
data: datasets/hagrid/data.yaml
epochs: 100
patience: 50
batch: 8
imgsz: 640
save: true
```

```
save_period: 2
cache: false
device: null
workers: 8
project: null
name: null
exist_ok: false
pretrained: false
optimizer: SGD
verbose: true
seed: 0
deterministic: true
single_cls: false
image_weights: false
rect: false
cos_lr: false
close_mosaic: 10
resume: false
overlap_mask: true
mask_ratio: 4
dropout: 0.0
val: true
split: val
save_json: false
save_hybrid: false
conf: null
iou: 0.7
max_det: 300
half: false
dnn: false
plots: true
source: null
show: false
save_txt: false
save_conf: false
save_crop: false
hide_labels: false
hide_conf: false
vid_stride: 1
line_thickness: 3
visualize: false
augment: false
agnostic_nms: false
```

```
classes: null
retina_masks: false
boxes: true
format: torchscript
keras: false
optimize: false
int8: false
dynamic: false
simplify: false
opset: null
workspace: 4
nms: false
lr0: 0.01
lrf: 0.01
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1
box: 7.5
cls: 0.5
dfl: 1.5
fl_gamma: 0.0
label_smoothing: 0.0
nbs: 64
hsv_h: 0.015
hsv_s: 0.7
hsv_v: 0.4
degrees: 0.0
translate: 0.1
scale: 0.5
shear: 0.0
perspective: 0.0
flipud: 0.0
fliplr: 0.5
mosaic: 1.0
mixup: 0.0
copy_paste: 0.0
cfg: null
v5loader: false
tracker: botsort.yaml
save_dir: runs/detect/train12
```



NTNU

Norwegian University of
Science and Technology