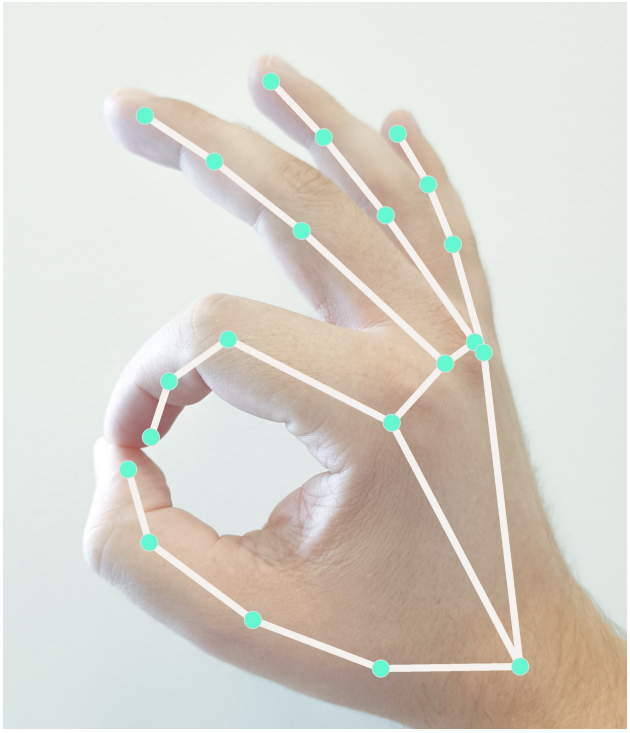

Hand gesture recognition in real time

Fast, accurate hand gesture recognition of static and dynamic hand gestures for on device classification

Degree project report in Computer Engineering

Jimmy Björklund

Department of Computer Science and Engineering

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Degree project report 2024

Hand gesture recognition in real time

Fast, accurate hand gesture recognition of static and dynamic hand gestures for on device classification

Jimmy Björklund

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024

Hand gesture recognition in real time
Fast, accurate hand gesture recognition of static and dynamic hand gestures for on device classification
Jimmy Björklund

Supervisor: Robert Krook, Computer Science and Engineering
Examiner: Jonas Almström Duregård, Computer Science and Engineering

Degree project report 2024
Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 72 731 2660

Cover: Visualization of skeleton data extracted by MediaPipe feature hand landmark detector.

Typeset in LaTeX
Gothenburg, Sweden 2024

Hand gesture recognition in real time
Fast, accurate hand gesture recognition of static and dynamic hand gestures for on device classification
Jimmy Björklund
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Human machine interaction (HMI) is an important part of the entertainment industry in that it allows developers to create an engaging experience. However when it comes to contactless interaction through hand gestures, it still remains a challenge to develop algorithms that are both accurate and fast enough to run in real-time on device [1]. In this report, this challenge have been explored using MediaPipe hand landmark detector as a feature extraction algorithm coupled with different classifiers trained to recognize both static and dynamic hand gestures. Result show the application can run both in real-time, and achieve relatively high accuracy of 88.1% for dynamic hand gestures and 93.1% on static hand gestures on large scale datasets. Furthermore, fine-tuning the dynamic hand gesture recognition algorithm to a specific user, was shown to improve the accuracy to 97.5%.

Keywords: hand gesture recognition, real time, MediaPipe, skeleton data, LSTM, self attention, dynamic hand gesture recognition, static hand gesture recognition.

Acknowledgements

This report was inspired by augmented reality applications, where one can interact with a software applications seamlessly in real-time without need for consoles. Much work has been put into making this project a reality, and a special thanks to my supervisor Robert Krook, for providing great support throughout this whole project.

Jimmy Björklund, Gothenburg, October 2024

Nomenclature

Below is the nomenclature of parameters and variables that have been used throughout this thesis.

Parameters

x_t	input feature vector at time step t
σ	sigmoid function
tanh	hyperbolic tangent function

Variables

C_t	cell state at time step t
$\tilde{C}_t$	candidate cell state at time step t
d_k	scaling factor for self attention
f_t	result of forget gate at time step t
h_t	hidden output at time step t
i_t	result of input gate at time step t
K	key matrix
o_t	result of output gate at time step t
$P(E)$	probability of event E occurring
$P(H\|E)$	probability of outcome H given event E occurred
$P(H)$	probability of outcome H
Q	query matrix
V	value matrix
W_{hc}	weight matrix for cell state associated with hidden state h
W_{hf}	weight matrix for forget gate associated with hidden state h
W_{hi}	weight matrix for input gate associated with hidden state h

W_{ho}	weight matrix for output gate associated with hidden state h
W_k	weight matrix for calculating key matrix
W_q	weight matrix for calculating query matrix
W_v	weight matrix for calculating value matrix
W_{xc}	weight matrix for cell state associated with input x
W_{xf}	weight matrix for forget gate associated with input x
W_{xi}	weight matrix for input gate associated with input x
W_{xo}	weight matrix for output gate associated with input x

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

Human machine interaction is a broad research field including many methods of communication such as audio based ones like speaker recognition, sensor based ones like keyboard and mouse, as well as techniques based on vision like hand gesture recognition [4][5][6]. Furthermore, it is also possible to combine many of these techniques together in order to achieve an overall higher quality of communication between human and machine [6]. For instance, using hand gesture recognition as an alternative to a keyboard can offer advantages such as being almost completely silent. Additionally, it can also allow for a more seamless experience when used with a graphical element since the user is not forced to switch their view between the screen and keyboard. When used for things like gaming, hand gesture recognition can potentially also offer for a more fun and engaging experience. However, when it comes to implementation of hand gesture recognition there are still several challenges left to be solved with regards to classification accuracy and the rate at which classifications can be made[1]. Part of the reason why achieving high classification accuracy is so difficult is due to the fact that the hand is relatively small, and often ends up being occluded[7] (i.e. by objects it may interact with or the other hand). Therefore, developing algorithms that can balance achieving high classification accuracy while keeping the complexity of the algorithm low enough to enable fast predictions is still a challenge for current state of the art algorithms [1].

1.2 Purpose

In the very recent years, there have been significant rapid advancements in the field of AI and machine learning, such as large language models like Chat GPT, AI generated images like those generated by Midjourney and much more. From this perspective it is therefore interesting to investigate how problems such as human-machine interaction potentially could be improved by better machine learning models. One interesting problem within this area is the challenge of implementing hand gesture recognition that is both fast and accurate, as hand gesture recognition represents a silent, intuitive and contactless interaction with many potential real world applications such as gaming and navigation. This project will try to address this challenge, specifically using RGB cameras as its sole sensor technology. Part of the motivation for this is that cameras represent a mature sensor technology commercially available at relatively low cost, which means the process of finding and collecting datasets is

simplified. Additionally, working with mature sensor technologies also has the benefit of making it more easily integrated into practical applications in the near future. While the project is aimed at developing the machine learning model for hand gesture recognition and analyzing its performance in terms of accuracy and speed, the project will also be somewhat research focused in nature. Specifically, it will include analysis of issues and bottlenecks found to limit performance in order to aid further research into the area.

1.3 Goals

The goal of the project is to develop a system capable of hand gesture recognition in real time including a small demo application for testing it in a live setting. Hand gestures to be recognized include both dynamic gestures such as swiping motions to the left or the right, as well as static hand gestures like rock sign (i.e. closed fist), ok sign, and palm sign.

Specifically, the questions to answer are:

1. What kind of accuracy can be achieved on static and dynamic hand gestures respectively when balancing this with real-time performance requirements?
2. What kind of real-time performance can be achieved in terms of latency?
3. What are some of the issues and bottlenecks in the system in order to achieve even greater performance in regards to speed and accuracy?

1.4 Limitations / Demarcations

Due to time restrictions, the project will be developed in Python in order to keep focus on the machine learning aspect of it. Additionally, due to the time consuming process of collecting datasets, only a small scale dataset of a single person will be used to evaluate how the model generalizes to the real world. With regards to the demo application used for measuring the real-time performance of it, the application will consist of a live video with the predicted class class shown in text form.

2

Theory

In principle, hand gesture recognition can be divided into two main tasks; feature extraction and classification [8]. The purpose of feature extraction being to reduce the raw input containing low level information into useful higher level semantic information such as skeleton data, see Figure 2.1 [8]. One such feature extraction algorithm capable of running in real-time, is the MediaPipe hand landmark detector, which will be discussed in more detail below [2]. With regards to selecting a classifier for hand gestures, the type of model to use depends on what type of hand gestures are to be classified, as static hand gestures are characterized by a single image, whereas dynamic hand gestures are characterized by a sequence of images. As such, dynamic hand gesture recognition models are required to handle sequential input, whereas static hand gesture recognition models allow for simpler models such as decision trees, random forest or feed forward neural networks (FFNN) to be used instead. In this report, models suited for dynamic hand gesture recognition have been explored using both LSTM and self attention. Details of how these models work are included in section 2.2. Finally, a section has also been dedicated to evaluation metrics for classification performance (see section 2.4).

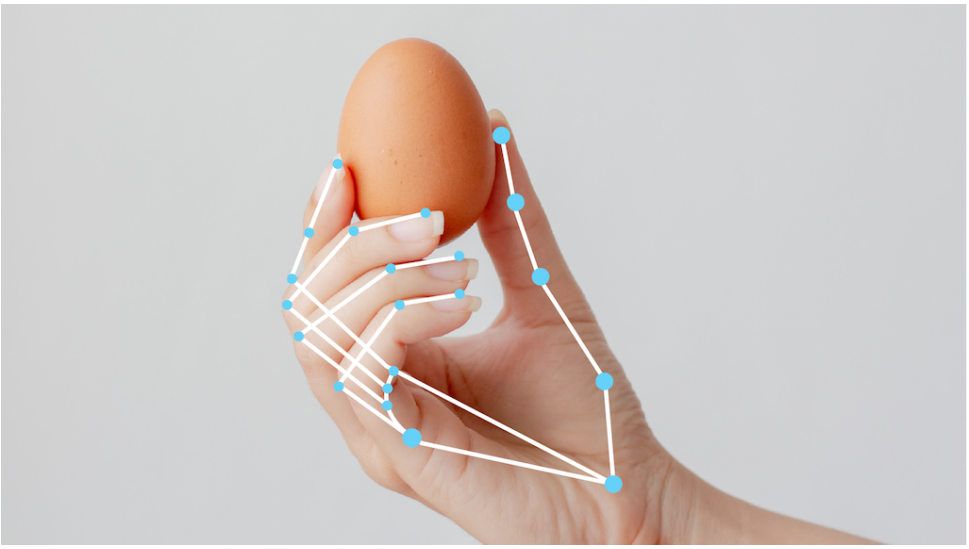

Figure 2.1: Example image showing skeleton data for hand [2].

2.1 MediaPipe hand landmark detector

In theory, feature extraction for hand gesture recognition could be performed in many different ways, however one well established method for feature extraction with reported high levels of accuracy when combined with a classifier is to produce skeleton data [9]. This is also the approach taken by the MediaPipe hand landmark detector (see Figure 2.2) [2]. What distinguishes MediaPipe hand landmark detector from many other solutions, is the speed at which it is able to do so, with benchmark measurements for average latency achieved at as low as 17.12ms with CPU and 12.27ms with GPU on a Pixel 6 phone [2] [10]. Part of the reason it is able to achieve so fast performance, is due to it architectural design, which includes a two part model that operates on the full image, and essentially provides a cropped image to the second model whose purpose is to define the location of the hand landmarks [2]. Furthermore, using the hand landmarks to define a bounding box for the hand and applying this to subsequent images, the need to run the relatively time-consuming palm detection model is completely eliminated until such an event a hand can no longer be detected in the bounding box [2]. With regards to output of the model i.e. skeleton data, it may seem counter intuitive that a model can produce 3D coordinates from a single image. However by performing some of its training using 3D hand models with known x,y, z coordinates rendered on different backgrounds, inferring even the z-dimensions was possible[10]. Additionally, the MediaPipe hand landmark detector has also achieved some robustness to partially visible hands as well as self occlusions, through training on this dataset as well as on an extensive real world dataset consisting of over 30K images [10] [2].

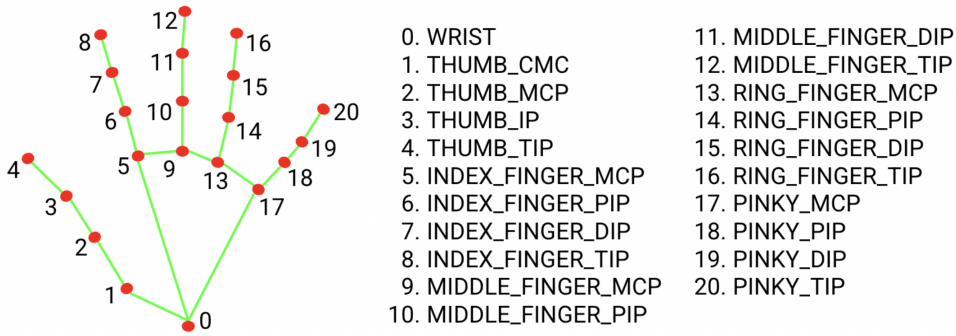

Figure 2.2: Definition of hand landmark points extracted by the MediaPipe hand landmark detector [2].

2.2 LSTM

Long-short-term-memory, or as it is most commonly referred to: LSTM is a form of recurrent neural network (RNN) [11]. Unlike regular feed forward neural networks (FFNN), recurrent neural networks are designed with loops, essentially allowing them to connect information captured from previous inputs with the current input (see Figure 2.3) [11]. Because of this memory like functionality, LSTM often work very well for problems such as sequence classification where capturing temporal

patterns are key [12] [13]. However when it comes to sequence classification tasks it is worth to note that not every output from each time step may be worth forwarding to the next layer of the model. In fact simply forwarding the output from the final time step to a classifier is often sufficient.

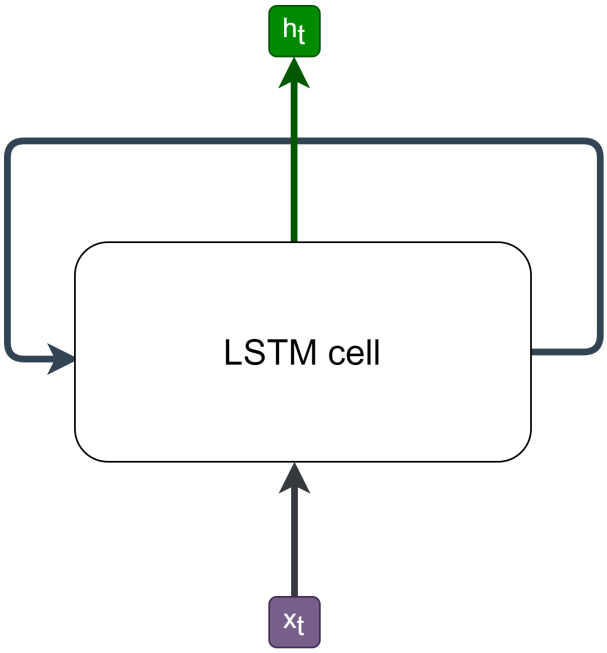

Figure 2.3: Schematic overview of LSTM.

2.2.1 Technical description of the LSTM cell

In order to understand how the LSTM cell works from a more technical point of view, the cell state must first be explained. The cell state, is essentially the memory of the LSTM cell and can be updated by both adding and removing information using mechanisms called gates [13][14]. The forget gate, as the name implies is responsible for selecting which information should be removed. How this is achieved in practice is with a bit of matrix algebra and a sigmoid function as shown in equation (2.1) [15]. While this equation may look a bit intimidating, it can basically be broken down into two important steps, the first being the matrix algebra inside the sigmoid function, and second, the sigmoid function itself. The purpose of the matrix algebra being to enable using learnable parameters (W_{hf}, W_{xf} and b_f) to determine what elements gets selected for removal based on information from the input features of the current time step (x_t) and the output of the previous time step (h_{t-1}). The purpose of the sigmoid function on the other hand being to make sure the output of the forget gate f_t are values between zero and one [14]. By performing this scaling, and using the forget gate in subsequent matrix multiplication with the cell state (shown later), values of zero will essentially represent completely forgetting information, whereas a value of one will represent completely retaining that information [14].

$$f_t = \sigma((W_{hf} \times h_{t-1}) + (W_{xf} \times x_t) + b_f) \tag{2.1}$$

Adding information is done using the input gate, which is performed using the equations: (2.2) - (2.4). The purpose of equation (2.2) being to decide which values to update, whereas the purpose of equation (2.3) is to define what information to add (often referred to as the candidate memory state) [14][15]. Similar to equation (2.1), both of these equations uses similar matrix algebra except with different learnable parameters to define its output. Finally in equation (2.4), old information from the previous cell state gets removed by multiplication with the forget gate (2.1), and new information gets added using simple addition of the selected memory defined by the outputs of equations (2.2) and (2.3) [14] [15].

$$i_t = \sigma((W_{hi} \times h_{t-1}) + (W_{xi} \times x_t) + b_i) \tag{2.2}$$

$$\tilde{C}_t = \tanh((W_{hc} \times h_{t-1}) + (W_{ci} \times x_t) + b_c) \tag{2.3}$$

$$C_t = (f_t \times C_{t-1}) + (i_t \times \tilde{C}_t) \tag{2.4}$$

Finally, all that remains is to explain is how the cell state affects the output of the LSTM cell. The math for this consists of these two equations (2.5) and (2.6), where the purpose of the first is a create a vector for selecting what parts of the cell state to output, and the second to push the values of the cell state between -1 and 1 using the tanh function before outputting the selected values [14][15].

$$o_t = \sigma((W_{ho} \times h_{t-1}) + (W_{xo} \times x_t) + b_o) \tag{2.5}$$

$$h_t = o_t \times \tanh(C_t) \tag{2.6}$$

2.2.2 Training LSTM on variable length input data

A consequence of the loop architecture that the LSTM is built upon, is that it consists of a single set of parameters that are shared for all time steps [13]. Because of this, LSTM models, are capable of handling variable length input [12]. However training a LSTM model can still be challenging when the data comes in different input lengths, as training LSTM:s in mini-batches, a commonly used technique in machine learning[16], requires handling the input data in a matrices which is somewhat problematic when the dimensions of the input is not consistent. Specifically by storing shorter sequences in a larger matrix, the challenge becomes how to prevent the model training process from accidentally using these last non-relevant elements and introduce errors as a consequence of it. One way around this issue is a strategy called bucketing i.e. a sampling strategy that only picks samples of the same length to be used in the same mini-batch. The advantage of this strategy being it is relatively easy to implement, while the disadvantage being that if the dataset is small or very diverse, may not have enough samples to fill every mini-batch for each different sequence length, and as such discard potentially important data for the model to learn on. Alternatively, there is also a strategy centered around padding the shorter sequences with a padding token, and using a masking process to keep this data from being processed [17]. Lastly there is also a third strategy essentially

combining both of them (see Figure 2.4). By grouping samples of approximately the same length, and then using padding and masking to solve for the small differences in input length the added advantage is reduced memory requirements during training, since comparatively less padding is used.

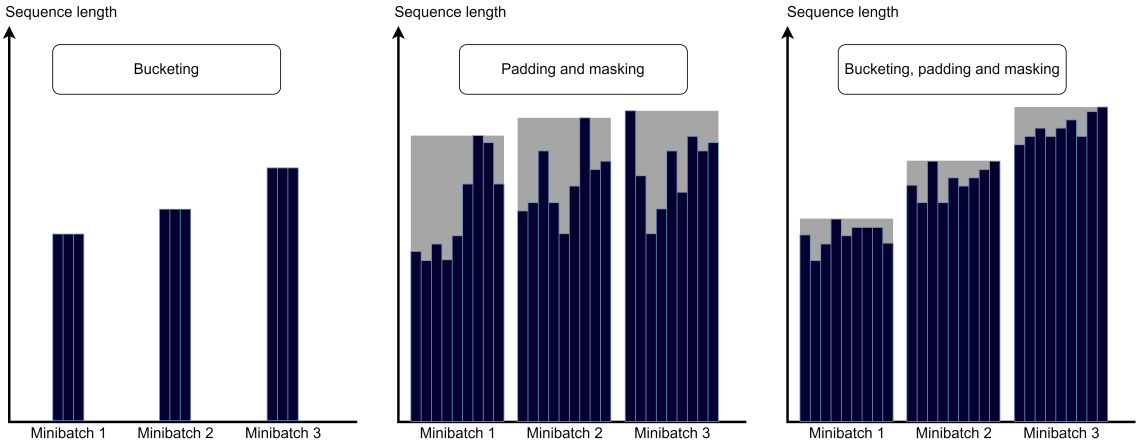

Figure 2.4: Different strategies for handling variable length input with minibatches. Gray color indicates padding.

2.3 Self attention

Recurrent neural networks such as LSTM:s have long been considered state of the art when it comes to sequence modelling tasks, such as sequence classification [18] [11]. However with recent advances, self attention is now replacing the need for recurrent neural networks like LSTM:s in certain applications [13]. Perhaps most notably in the transformer design first introduced in the paper "Attention is all you need" 2017, which has since become a critical component of the famous Chat GPT model [18][19]. Part of the reason for its success is its ability to capture relationships between elements in an input sequence regardless of distance between them [18]. A somewhat simplified explanation of how this mechanism works is by identifying, and weighing the importance of the different elements of the input sequence, and mapping that information to an output [18]. Thus self attention very effectively enables capturing a contextual understanding of the relationships within the input sequence [18].

2.3.1 Technical description of how self attention works

A more mathematical explanation of how self attention (scaled dot-product attention specifically) works involves taking the input sequence (X) and multiplying it with the three different weight matrices according to the equations in (2.7) to produce the query, key and value matrices [20]. These matrices are then used in equation (2.8) to produce the output.

$$\begin{aligned} Q &= W_q X \\ K &= W_k X \\ V &= W_v X \end{aligned} \tag{2.7}$$

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.8}$$

To understand why this might make a bit of sense, lets start by analyzing the matrix multiplication of the query matrix with the transposed key matrix. As this operation is equivalent to a dot product of the query with all the keys, and a dot product is a measure of similarity, the operation results in numbers reflecting which keys are more relevant than others relative to the specific query (i.e. input element)[18] [20]. This becomes important since the purpose of the keys, similar to a common dictionary data structure, is to provide a mapping of what to output. Next the result is scaled, first by a factor $\frac{1}{\sqrt{d_k}}$ (for optimization reasons) and then again by applying a softmax function [18]. Performing this scaling and doing the final matrix multiplication, the result i.e. the output becomes a weighted sum of the value matrix based on the importance of the keys. Thus by tuning the values of the weight matrices through back propagation, the output can become a representation that successfully captures the important relationships within the sequence.
Instead of relying on a single self attention mechanism, it is possible to run multiple self attention mechanisms in parallel, by linearly projecting the query, key and value matrix in different ways, to perform so called multi-headed attention. This has the potential benefit of being able to capture more different kinds of relationships in the input sequence [18].

2.4 Evaluation metrics for classification performance

Performance metrics for classification models can be quantified in many different ways, such as accuracy, precision, recall and many more. In practice it is common to use several of these metrics to analyze the model's performance. To understand why this is necessary, consider the example of a model with 99% accuracy. While the 99% at a glance would indicate the model performs well, that may not be the case if the dataset is composed of a class that makes up 99% of the dataset, since the model may simply predict that majority class every time. Essentially, this model fails to correctly predict any of the other classes, and thus the model would be useless. This is an obvious case where accuracy is known to produce misleading conclusions. However the fact remains, accuracy alone does not provide a good intuition on whether the model is struggling with any specific class. As such, including performance metrics like precision and recall enables a more comprehensive analysis. Specifically, what precision provides is a quantitative measure of how many of the of the predicted class that were actually of that class [21]. Similarly, recall provides a metric for how many of a class that were correctly classified as that class

[21]. If both recall and precision are high (i.e. near one), the model can be assumed to perform well. Formal definitions of these equations are shown in equations (2.9) and (2.10), where the false positives(FP), false negatives(FN), true positives(TP) and true negatives(TN) are extracted from the confusion matrix.

$$\text{precision} = \frac{TP}{(TP + FP)} \tag{2.9}$$

$$\text{recall} = \frac{TP}{(TP + FN)} \tag{2.10}$$

3

Methods

Designing a solution for a system meeting the requirements of being able to perform both static and dynamic hand gesture recognition in real time is a process consisting of many different steps. These steps, from a high level perspective include designing an overall model architecture, developing machine learning models and implementing a demo application. A more detailed description what each of those steps were are included below.

3.1 Model architecture

The proposed architecture for the full model can be seen in Figure 3.1. While at a glance, the design choice to have two classification models, may seem an odd choice, since a static hand gesture in theory could be considered a dynamic hand gesture with a sequence length of one. However, this would be problematic due to the fact that static hand gestures could be a subset of a dynamic hand gesture. For instance, the swiping down motion, essentially, could include the palm sign as one of its images in the downwards movement. Because of this, one would either have to find a dataset that properly accounts for this using proper labeling on every image in addition to every sequence or risk a performance loss associated with training on inaccurately labeled data. The proposed solution in this project solves this issue by training separate classification models, such that the static hand gestures are only trained on static images (i.e. sequence of length one) thus avoiding potential subsets altogether. However one issue that should to be considered when using multiple classification models in an architecture like this, is what the final output should be for each possible combination of output from the individual models. Specifically, it is easy to overlook the possibility that both models may predict a positive class (i.e. any class other than the no gesture class) simultaneously, and depending on the application, this could introduce unintended behaviour. In this report, the solution have been designed to utilize the dynamic gesture recognition result first, and only if it predicts a negative consider the result of the static gesture recognition classifier to remove the possibility of outputting two positive classes simultaneously.

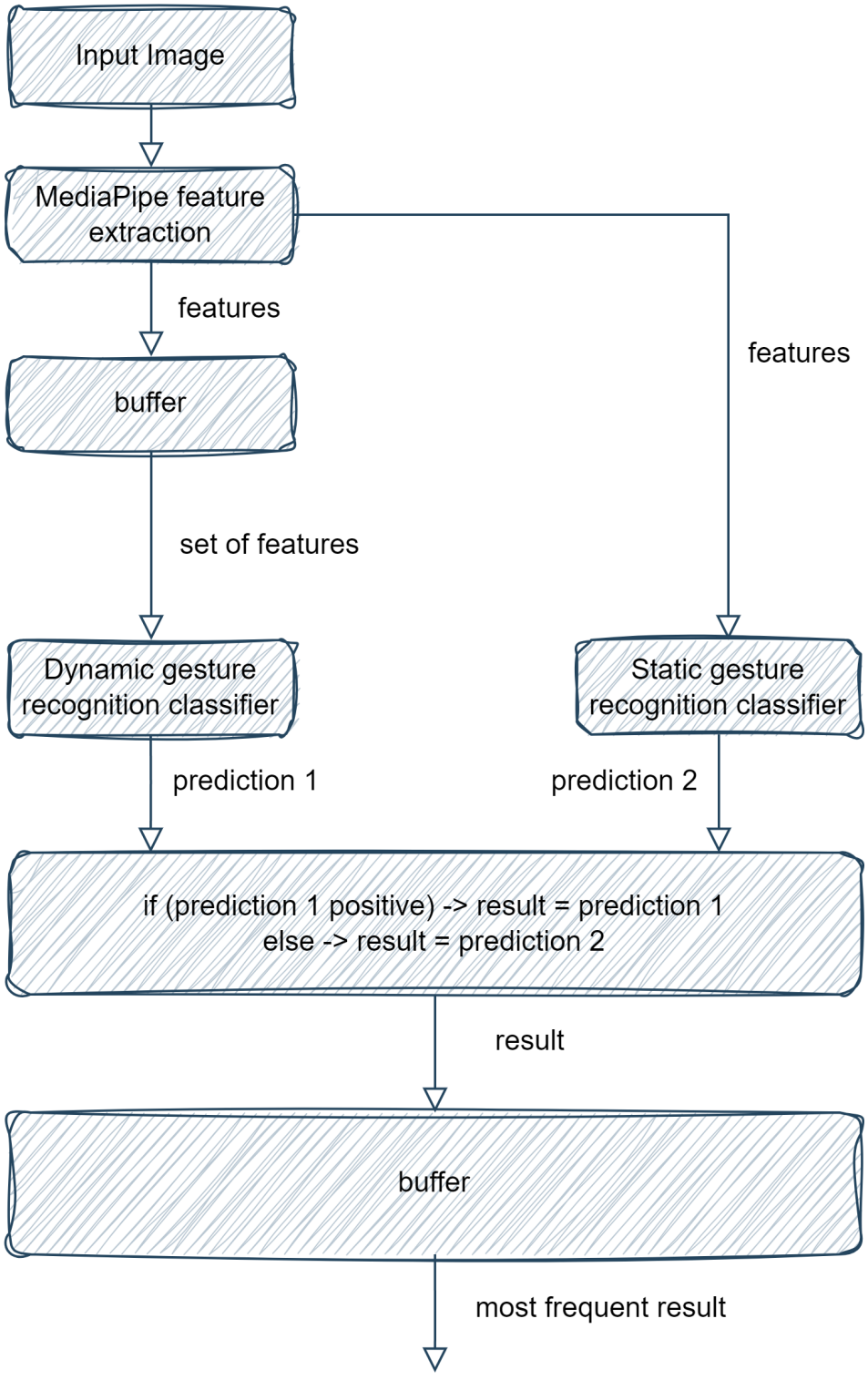

Figure 3.1: Schematic overview of the model architecture for the integrated system implementing both static and dynamic hand gesture recognition. As results are generated very frequently (i.e. every frame), the purpose of the final buffer is to provide the most reliable output by taking the most frequent prediction from a relatively small number of frames.

3.2 Dynamic hand gesture recognition

Development of this classification model was performed using the Jester dataset. This is a dataset collected using crowd sourcing with 1300 unique persons, using front facing cameras captured at 12 FPS [22]. Despite its low image size (100 px height), it has been used frequently in research literature as a benchmark for classification performance [22] [13]. In this project however, only a subset of this dataset is used, specifically the classes: swiping left, right, up and down in addition to the two negative classes labeled as "No gesture" and "Doing other things".

3.2.1 Dataset preparation

The Jester dataset was checked for both missing and duplicate labeling as well as mislabeled samples. However due to the large amount of data, only small subsets could be manually inspected for mislabeling. No mislabeling errors were found, however when manually inspecting images, it seemed clear that for some image sequences in the Jester dataset, that some images would be copies of its previous image in the sequence (example: id 10125, image 00007 and 00008). While interesting to note, no action was taken as direct a result of this, partly due to the fact that Jester dataset have previously been used for benchmarking classification performance in research papers [23] indicating a relatively high quality, and also due to the problematic nature of removing single images from a sequence would cause, since the sequence is supposed to be collected at a constant frame rate.
Due to the sheer size of the dataset, extracting hand landmark data using the MediaPipe algorithm was performed as a pre-calculation step to optimize the otherwise very long training time. The reasoning behind this decision was motivated by the fact that the MediaPipe algorithm was estimated capable of processing approximately 30 images per second sequentially one a single CPU in Python, which when considering that the subset of the Jester dataset in this project contains well over one million images, thus infer a training time well over 9 hours per epoch simply from the feature extraction process. Comparatively, by using pre-calculated features, training one epoch would take a few seconds in total depending on the specific model and its exact hyperparameters.
With regards to pre-processing steps performed, very little had to be done, since the MediaPipe feature extraction algorithm performs many of these operations internally such as scaling the input image, and normalizing the output relative to the image size. Hence the pre-processing mostly consisted of flattening the 21 handlandmarks per hand into a vector. To handle the case, where the MediaPipe algorithm may not detect both a right and left hand in the image, corresponding zeroes would be inserted to indicate this.

3.2.2 Model design

Models for dynamic hand gesture recognition were developed through an iterative process building on the bottom up principle i.e. only adding complexity after evaluating the simpler version of the model first. The first model trained for dynamic

hand gesture recognition was a LSTM model as these are well known for their performance for sequence classification tasks [18] [11]. Once a simple version of a LSTM model capable of training using mini-batches with the "bucketing" sampling strategy was implemented, some manual hyperparameter tuning was performed with these parameters: batch size, hidden size, learning rate, dropout rate, and number of LSTM layers stacked on top of each other.
Following this optimization step and a quick evaluation of performance, a visual inspection analysis was performed (see section 3.2.3) in order to gain insight into weaknesses of the model, and implement potential solutions accordingly. Using this information, and performing some general minor improvements, such as updating the sampling strategy during the training process from a "bucketing" strategy to a combined bucketing/padding/masking strategy, 8 new models were developed. These 8 models are listed in Table 3.1, including a short description of its design. Of these 8 models, the first 5 of are small variations of the initial LSTM model, differentiating mostly in their hyper parameters, the number of linear layers, or regularization techniques employed. Model 6, is also a variation of the LSTM model, except combined with a multi-headed self attention mechanism, included in an attempt to improve capture of long range dependencies. Following this model, two variations using multi-headed attention were also developed without the LSTM layer, primarily differing in how many heads were used to capture different patterns in the data.
As a final optimization step to try to minimize overfitting, training the same 8 models using an augmented version of the Jester dataset was also performed. The augmented data added to only the training part of the original dataset consisted of small random transformations in brightness, saturation, hue, sharpness and contrast. The exact values for these settings were found through an iterative process of trying to manually adjust the settings to produce small, but noticeable differences, and then verify that the difference between the number of hands detected by the MediaPipe algorithm before and after would not differ more than approximately 10%. This extra step of verification was added, based on the assumption that losing information in the feature extraction process, could have negative consequences for the training process. With regards to randomization, the decision was taken to apply different transformations to all sequences, but the exact same transformation for all images within a sequence. This decision, while possibly not very important, was motivated by the fact that the transformations, reassemble using different camera settings and as such would logically remain constant through the capture of an image sequence. The details of these models trained on augmented datasets are listed in Table 3.2.

Table 3.1: Dynamic hand gesture recognition models trained on Jester dataset. Of the models listed, only model 2 have been automatically optimized using an Bayesian optimization algorithm with these parameters: number of LSTM layers, batch size, learning rate and hidden size [3]. Note LL here is short for linear layers.

model alias	model description
1	1 LSTM layer, 1 linear layer
2	2 LSTM layers, 1 LL, bayesian optimized parameters
3	3 LSTM layers, 1 LL
4	3 LSTM layers, 2 LL
5	3 LSTM layers, 2 LL, layer norm
6	3 LSTM, 8 self attention heads, 2 LL, layer norm
7	2 self attention heads, 2 LL, layer norm
8	8 self attention heads, 2 LL, layer norm

Table 3.2: Dynamic hand gesture recognition models trained on an augmented version of the Jester dataset.

model alias	model description
9	same as model 1 but trained on augmented dataset
10	same as model 2 but trained on augmented dataset
11	same as model 3 but trained on augmented dataset
12	same as model 4 but trained on augmented dataset
13	same as model 5 but trained on augmented dataset
14	same as model 6 but trained on augmented dataset
15	same as model 7 but trained on augmented dataset
16	same as model 8 but trained on augmented dataset

3.2.3 Visual inspection analysis

Visual inspection was performed using a sampling process consisting of applying an early version of model 3 (utilizing bucketing sampling instead of the bucketing, padding and masking strategy) and randomly sampling 10 sequences from each group of the confusion matrix evaluated on the training part the Jester dataset. In those instances 10 samples (i.e. image sequences) would not exist, fewer samples would be taken from that group. The basis for this sampling process was a desire to get enough samples to enable investigating very specific types of miss classifications such as confusing swiping left with swiping right. In the end, processing these samples with the MediaPipe's output i.e. hand landmarks drawn on top of the image, required manually inspecting over 11 000 images, which proved quite time consuming. Thus the inspection was mostly limited to annotating general comments, and if the MediaPipe algorithm had performed a false positive i.e. detected a hand where no hand really existed. The processing of this visual inspection data was then performed using Bayes Theorem to estimate the risk factor of having a false positive in one or more images in an image sequence. Additionally, the frequency of occurrence of these so called false positives were also calculated and used with the

previously mentioned risk factor to calculate an expected number of errors caused by false positives under the assumption of a cause effect relationship. While the cause effect relationship assumption was not investigated itself, this analysis was used as an indicator to understand the potential magnitude of the problem associated with false positives, and if effort should be taken to find solutions to work around this very specific problem.
In combination with the visual analysis, a second, completely automated analysis was also performed. Specifically by calculating the average number of hands detected for each group in the confusion matrix, with the aim to find any correlation between low detection rate by the MediaPipe algorithm and a resulting missclassification.

3.2.4 Dataset collection

To investigate how the best performing model on the Jester dataset generalizes or could be adapted to the real world, a dataset consisting of 600 image sequences i.e. 100 of each dynamic hand gesture class was collected (see details in Table 3.3). This data was then used for these three types of evaluations:

1. Evaluate the classification accuracy of the best performing model on the Jester dataset performs on this dataset.
2. Evaluate how higher capture rates by the webcam improves or degrades the classification accuracy.
3. Evaluate how fine tuning the best model on the 80% (70% training + 10% validation) of the data improves classification performance on the remaining unseen 20%.

The protocol used for collecting the image sequences consisted of starting with inspecting a few random examples from the Jester dataset for each class. This was motivated by the fact that original collection protocol used in the Jester dataset could not be found, and that even simple gestures to some degree are open to interpretation. For instance, swiping gestures could be small, or large, include different amount of arm or wrist movement, or even start from different positions such as free hanging arms or in a more ready like position. Once complete, a bit of practice was performed in order to make the movements feel more natural. Collection of data was then performed in batches of 20 of all six classes, using a script where each capture sequence would begin with a 7 second timer counting down to start, and ending by pressing a keyboard button with my left hand (since that hand was free and out of camera). In between each collection batch, different lighting conditions would be arranged to include both bright backlit conditions, and relatively darker conditions. Furthermore, different clothing and items in the background were used in order to introduce some small amount diversity into the dataset.

Table 3.3: Hardware and software settings used during data collection.

Component	
Webcam model	Logitech C922 Pro Stream Webcam
Image resolution	1600 x 896
Capture rate	30
Software	Python 3.11 with OpenCV version 4.9.0

3.3 Static hand gesture recognition

Development of the classifier for static hand gesture recognition was primarily performed on the dataset: HAnd Gesture Recognition Image Dataset (or Hagrid for short). Similar to the Jester dataset, this is also a large scale, diverse, dataset collected through crowd sourcing. In total, 37 583 individuals were involved in the collection process of the 18 static hand classes using varying natural lighting conditions and camera distances of 0.5 to 4m. For this report, only the three classes: "palm", "ok" and "closed fist" were used totaling 94 406 images. The images while originally available at a size of: 1920x1080, were also available in a smaller resolution where the smallest dimension is constrained to 512px, which was the version used in this report. With regards to using a negative class, no usable negative class in the Hagrid dataset were found, and as such the decision to reuse the negative classes: "no gesture" and "doing other things" from the Jester dataset was taken. However as the Jester dataset is a much larger dataset when comparing number of images, only 32000 (or 54400 if the Hagrid dataset was augmented) from the negative classes were selected to be included in the training and evaluation of the static hand gesture classifier to maintain a balanced dataset.

3.3.1 Dataset preparation

Data preparation for the static hand gesture recognition was performed the same way as described in section 3.2.1 for the Jester dataset part, and almost the same way for the Hagrid dataset part as well, except nothing noteworthy was found during data validation. However instead of using the official annotations from the Hagrid dataset, labels were created manually, by reading the subfolder names of where the images were stored. This was motivated by the fact that the Hagrid dataset annotations consists of bounding boxes of the hands in the images, which is more complex in-depth information than what is required for this classifier.

3.3.2 Model design

Unlike the case for dynamic hand gesture recognition, which is a sequence to one classification problem, static hand gesture recognition is a relatively simpler classification problem [24] as it does not require capturing difficult sequential patterns [24]. In this project, several models from sk-learn library have been tried, as well as customized neural networks developed using the Pytroch library in Python. A list of these models and a short description of them can be seen in Table 3.4. With regards to hyperparameter tuning of these, all of it have been performed manually.

Table 3.4: Description of static hand gesture models and their alias.

model alias	model description
1	Neural network, 3 linear layers
2	Neural network (augmented), 3 linear layers
3	Neural network, 3 linear layers, dropout
4	Neural network (augmented), 3 linear layers, dropout
5	sk-learn: MLP, 2 layers
6	sk-learn: Random Forest, depth = 50, estimators = 200
7	sk-learn: Decision Tree, depth = 32
8	sk-learn: K-nearest-neighbors, k = 3

3.4 Performance evaluation metrics

Performance of the model, was estimated both in terms of speed and classification accuracy. The exact methods used for measuring these qualities are described in the subsections below.

3.4.1 Classification Performance

The classification performance for both dynamic and static hand gesture recognition models was estimated using a combination of accuracy, precision and recall and confusion matrix. Accuracy was selected, because of its simple intuitive meaning and because Jester, Hagrid and my own dataset have a very balanced distribution of classes. However in order to provide an in depth analysis of how the model performs for each different type of class, precision and recall were also selected to be used. Finally analysis of the raw confusion were used in some cases, to provide information on what kind of errors the model makes.

3.4.2 Execution speed

The real-time performance for the application and its subsystems (i.e. classification models) was measured as the average processing time per captured frame over a duration of 1000 frames. In total, five unique types of measurements were performed, one for the static and dynamic hand gesture classifiers each, as well as one for the overall model including steps such as formatting of input data, running both classifiers, and processing their individual predictions to a final output. In addition to these three measurements, one was also performed for the MediaPipe hand landmark detector as well one for the overall application including all its subsystems. Measurements for these processes were performed using the timeit package in Python and reported in milliseconds (ms). Note that the static and dynamic hand gesture recognition classifiers were run on the GPU, whereas all other parts of the code was run on the processor. Also, the static and dynamic hand gesture recognition models were run sequentially in the application. As the time measurements described are inherently dependent on both hardware and software, those details are specified in Table 3.5.

Table 3.5: Hardware and software specification used in real-time performance measurements

Component	
CPU	AMD Ryzen 5 3600 6-Core Processor 3.60 GHz
GPU	NVIDIA GeForce GTX 2060 super
RAM	16,0 GB
Language	Python 3.11
Operating system	Windows 10
Webcam	Logitech C922 Pro Stream Webcam
Webcam resolution	1600 x 896
Webcam capture rate	30

4

Results

The result in this section is divided into three subsections. The first two: "Static hand gesture recognition" and "Dynamic hand gesture recognition" dealing primarily with the model training and evaluation process for their respective task, whereas the last subsection: "Integrated system" contains a detailed analysis of the overall real time performance.

4.1 Static hand gesture recognition

This section deals with classification performance of static hand gestures. The classes to be recognized include: "palm", "closed fist" and "ok" signs in addition to the negative classes: "Doing other things" and "No gesture". Performance comparison of the models can be observed in Table 4.1. Additionally a confusion matrix for the best scoring model (i.e. model 1 with 93,2% accuracy) was evaluated on the test part of the Jester/Hagrid dataset and can be observed in Figure 4.1. A corresponding table of precision and recall values is also included in Table 4.2. Lastly, an impact analysis of MediaPipe's feature extraction process is summarized in Table 4.3.

4.1.1 Evaluation on Hagrid and Jester dataset

Table 4.1: Classification accuracy of models performed on the validation part of the Hagrid/Jester dataset.

Model alias	Algorithm description	Accuracy (%)
1	Neural network	93.2
2	Neural network (augmented)	93.1
3	Neural network	93.0
4	Neural network (augmented)	93.1
5	MLP	92.9
6	Random Forest	89.4
7	Decision Tree	88.3
8	K-nearest-neighbors	71.1

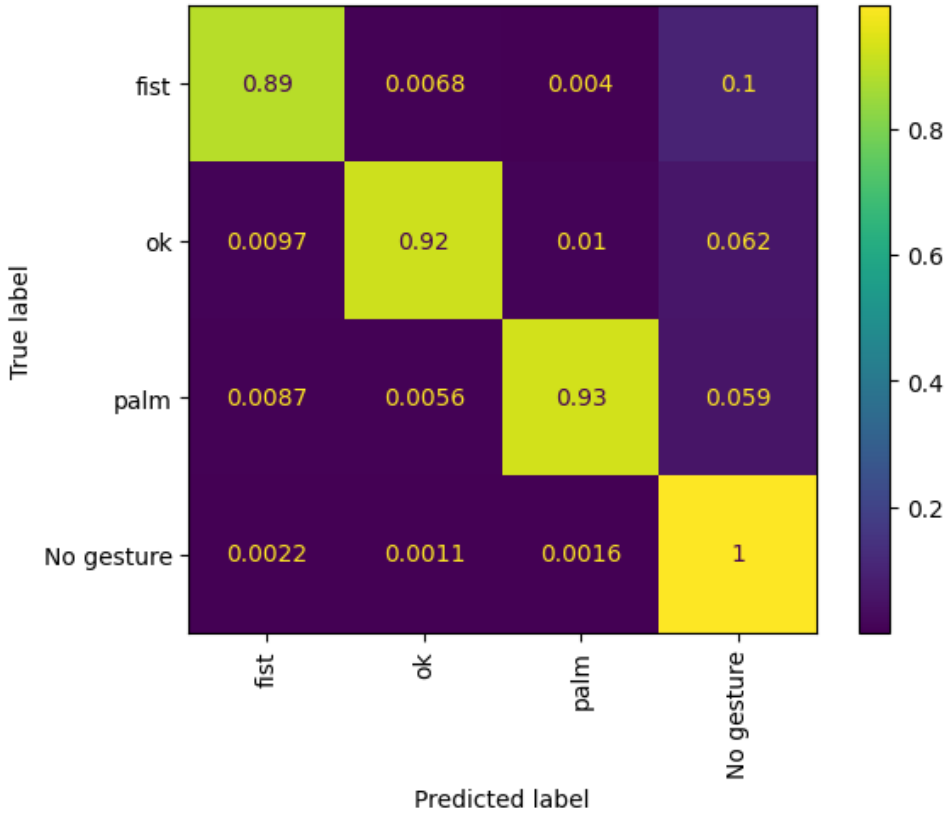

Figure 4.1: Normalized confusion matrix performed using model 1 on test part of Hagrid/Jester dataset. Accuracy: 93,2%.

Table 4.2: Class level precision and recall values for model 1 evaluated on test part of Hagrid/Jester dataset

Class	Precision	Recall
fist	0.98	0.89
ok	0.98	0.92
palm	0.98	0.93
No gesture	0.82	1.00

4.1.2 Analysis of MediaPipe feature extraction

Table 4.3: Error analysis of images with zero hands detected by the MediaPipe feature extraction algorithm. Note that the risk factor here is defined as the fraction of samples with zero hands detected that resulted in a false classification made by model 1 on the full non-augmented Hagrid dataset.

True class	% of all class errors	risk factor
fist	88.7	1.0
ok	76.2	1.0
palm	81.2	1.0

4.2 Dynamic hand gesture recognition

This section deals with classification performance of dynamic hand gestures. The classes to be recognized include: "Swiping left", "Swiping right", "Swiping up" and "Swiping down" in addition to the two negative classes: "Doing other things" and "No gesture" but labeled as a single negative class.

4.2.1 Evaluation on Jester Dataset

A performance comparison of the models developed on the Jester dataset can be observed in tables 4.4 and 4.5 for models trained on the original and augmented version of the dataset respectively. Evaluation of performance with model 11 on this test part of the Jester dataset resulted in an accuracy of 88,1% for model 11. Precision and recall values for this evaluation are also shown in Table 4.6, as well as a normalized confusion matrix in Figure 4.2

Table 4.4: Validation accuracy performed on Jester dataset using models trained without data augmentation.

model alias	accuracy (%)
1	87.1
2	87.1
3	87.7
4	87.6
5	87.4
6	87.5
7	83.8
8	84.5

Table 4.5: Validation accuracy performed on Jester dataset using the models trained on augmented datasets of Jester.

model alias	accuracy (%)
9	86.6
10	87.6
11	88.0
12	87.7
13	87.9
14	87.4
15	83.7
16	84.0

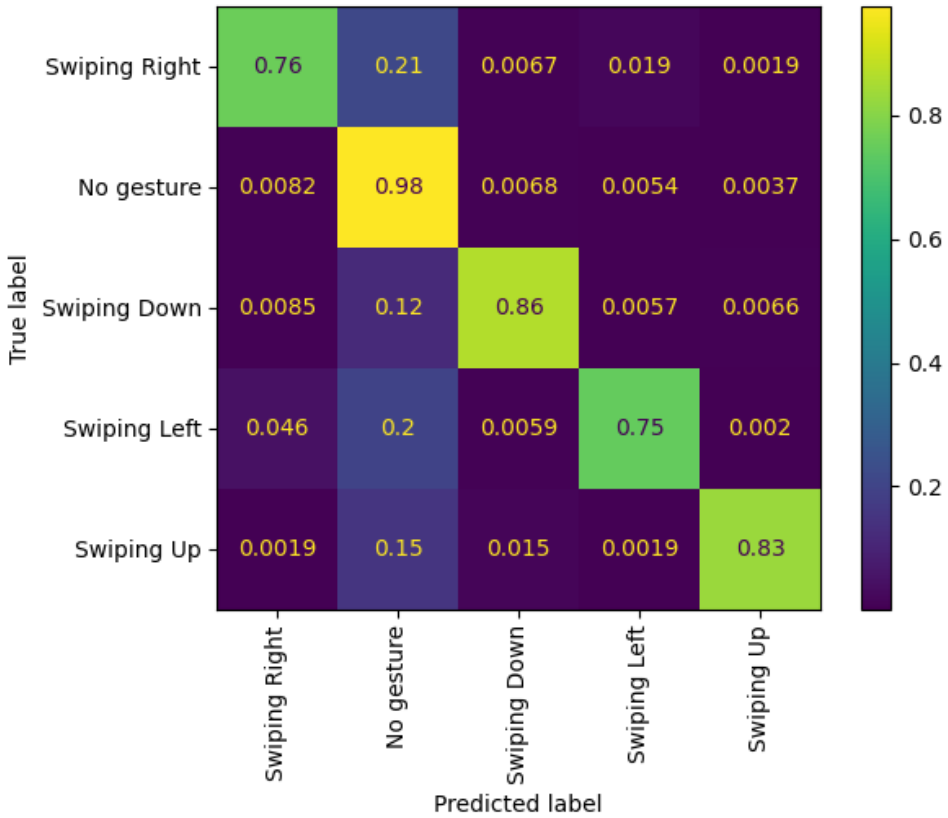

Figure 4.2: Normalized confusion matrix performed using model 11 on test part of Jester dataset. Accuracy: 88,1%

Table 4.6: Class level precision recall values for model 11 evaluated on test part of Jester dataset

Class	Precision	Recall
Swiping Right	0.90	0.76
No gesture	0.83	0.98
Swiping Down	0.94	0.86
Swiping Left	0.94	0.75
Swiping Up	0.97	0.83

4.2.2 Visual inspection analysis

By manually inspecting and annotating over 300 image sequences (i.e. $> 11\ 000$) images, several interesting observations could be made. For instance, many of the images, had very large amounts of motion blur, sometimes making even manual visual inspection a challenge. Furthermore, relatively frequently false positives, would occur in one or more images of a sequence. The false positives, would often manifest as confusing other body parts like a shoulder or face as a hand. A deeper analysis of both failing to identify hands (i.e. false negatives) and falsely finding hands where none exist (i.e. false positives) are shown in the subsections below. Finally, with regards to lighting conditions, examples of both very strongly lit, and very dark rooms could be found, but were not analyzed through any quantitative measure.

4.2.2.1 Analysis of false negatives from feature extraction step

A table comparing the average fraction of images with at least hand detected by the MediaPipe algorithm can be observed in Table 4.7.

Table 4.7: Average number of hands detected per image by the MediaPipe feature extraction algorithm. Note that the true average is unknown since hand gestures for example could start or end with the hand outside the cameras field of view.

True class	Correct prediction	Falsely predicting "No Gesture"
No gesture	0.2	NA
Swiping Left	0.48	0.052
Swiping Right	0.3	0.12
Swiping Down	0.34	0.05
Swiping Up	0.28	0.075

4.2.2.2 Analysis of false positives from feature extraction step

The impact on classification performance from MediaPipe making one or more false positives on the image sequence was analyzed using Bayesian statistics as shown below.

$H(+)$: final classification is false
$H(-)$: final classification is correct

E: Image sequence contains at least one image with a false positive made by the MediaPipe algorithm during feature extraction. One example of this is if a hand is detected on background clutter instead of a real hand.

Applying Bayes theorem, with the relvant probabilities extracted from the visual analysis, $P(H|E)$ can be calculated as shown in equation (4.1)

$P(E|H+) = 0.288$
$P(E|H-) = 0.163$
$P(H+) = 0.122$
$P(H-) = 0.878$

Bayes Theorem:

$$P(H|E) = \frac{P(E|H+) \cdot P(H+)}{(P(H+) \cdot (E|H+) + P(H-) \cdot P(E|H-))} = 0.197 \tag{4.1}$$

One way to interpret this number is that the risk of the model making a false prediction is 19,7% if the image sequence contains a false positive made by the MediaPipe algorithm. Considering the prior probability, when no information about MediaPipe's feature extraction is known is only 12,2%, this indicates the risk factor has almost doubled. However to properly quantify the risk false positives have on the model's classification performance, it is also important to factor in the frequency of occurrence. For instance, if false positives are very rare in the Jester dataset, most of the model's error may not be related to false positives at all, thus making it only a minor issue. The frequency of occurrence for image sequences containing one or more false positives are given by the formula:

$$P(E) = P(E|H+) \cdot P(H+) + P(E|H-) \cdot P(H-) = 0.17825 \tag{4.2}$$

Solving equation (4.2) yields $P(E) = 0.17825$, which essentially is an estimate that approximately 17.825% of the image sequences in the Jester dataset contains one or more false positives by the MediaPipe algorithm. To put this number in relation, one can multiply it with the risk factor. If one can assume a cause-effect relationship, then this product will equal the expected fraction of the whole dataset that are errors that can be attributed to the false positives. Since $0.17825 \cdot 0.197 = 0.035$, this number accounts for about 28,9% of the 12,2% errors the model makes. While a direct cause effect relationship have not been proven conclusively, this number is relatively large and as such could indicate a critical issue the model is struggling with.

4.2.3 Evaluation on personal dataset

A dataset with myself as test subject was collected to evaluate how well the model generalizes to the real world, and evaluate how fine-tuning potentially improve performance. In total, 100 unique sequences of each of the six dynamic hand gesture

classes were collected, where 70% was used for fine-tuning 10% for validation and 20% for testing performance as previously unseen data. During post collection inspection several examples of motion blur could be found such as shown in Figure 4.3.

Figure 4.3: A cropped image from my dataset displaying a case of motion blur.

4.2.3.1 Evaluation on personal dataset: 30 FPS

Evaluation of the performance on the dataset in its original 30 FPS capture rate was performed using model 11 and can be shown using a confusion matrix, as well as a precision recall table in Figure 4.4 and Table 4.8 respectively. The accuracy estimated on the whole dataset in this evaluation was 78.0%.

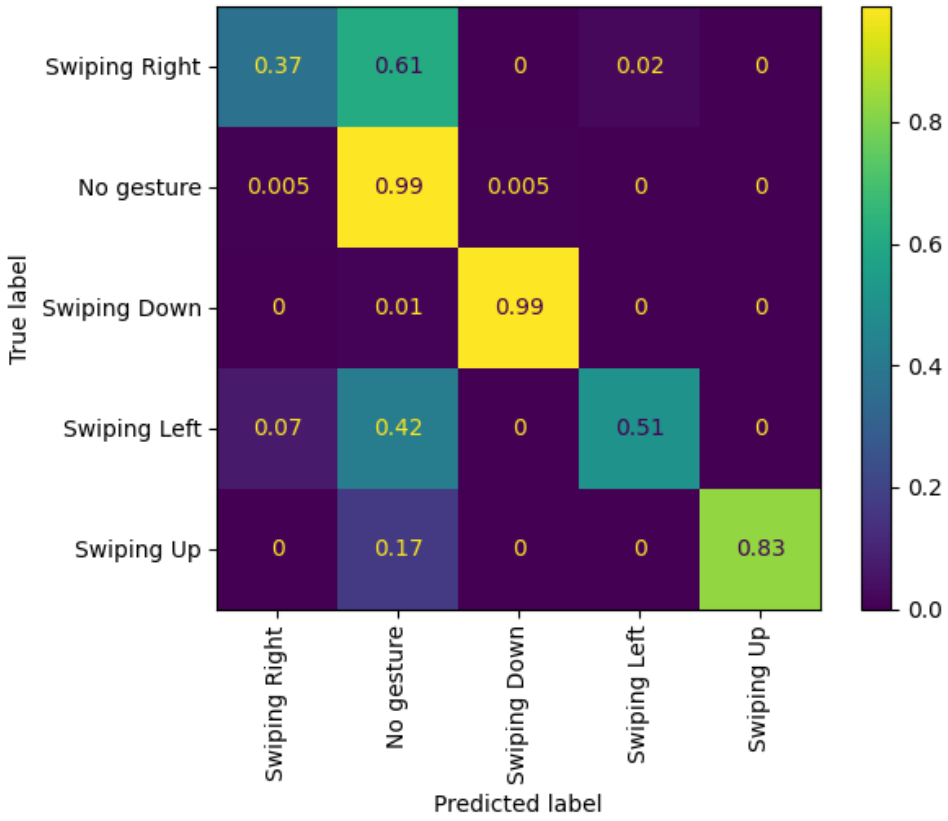

Figure 4.4: Normalized confusion matrix for model 11 applied on the whole personal dataset. Note that the dataset was captured at 30 FPS. Accuracy achieved: 78,0%.

Table 4.8: Class level precision recall values for model 11 applied on personal dataset. Note that the dataset was captured at 30 FPS.

Class	Precision	Recall
Swiping Right	0.82	0.37
No gesture	0.62	0.99
Swiping Down	0.99	0.99
Swiping Left	0.96	0.51
Swiping Up	1.00	0.83

4.2.3.2 Evaluation on personal dataset: 12 FPS

Evaluation of the performance on the dataset adjusted to 12 FPS was performed using model 11 and can be shown using a confusion matrix, as well as a precision recall table in Figure 4.5 and Table 4.9 respectively. The accuracy estimated on the whole dataset in this evaluation was 83.7%.

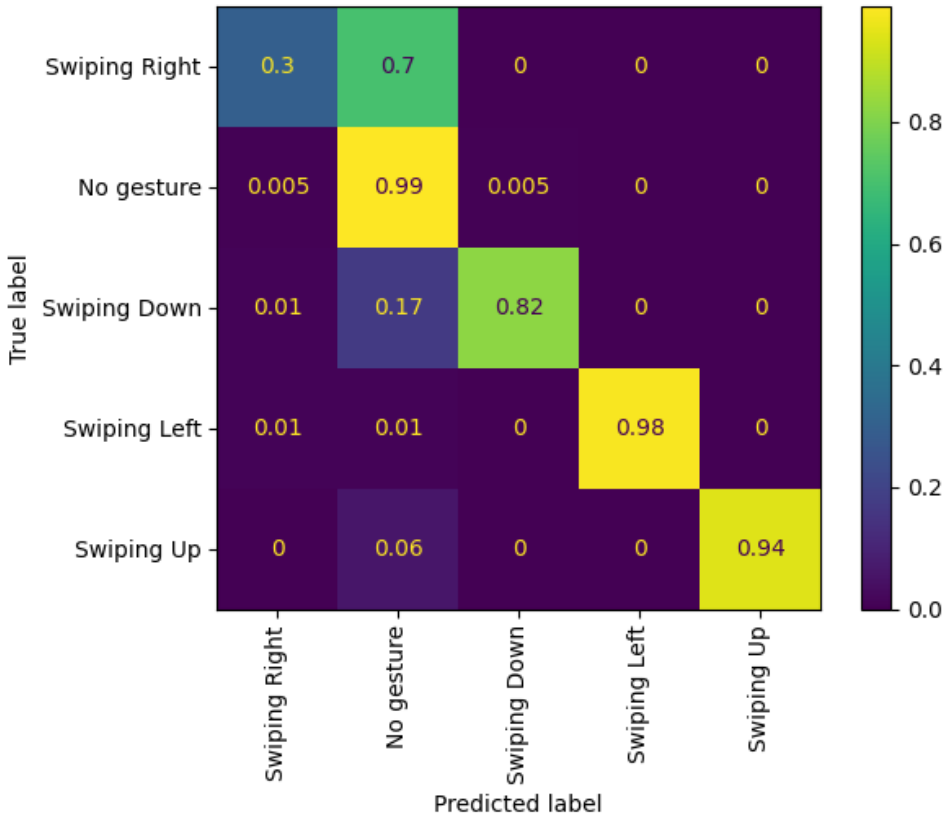

Figure 4.5: Normalized confusion matrix for model 11 applied on the whole personal dataset. Note that the dataset was converted to 12 FPS for this evaluation. Accuracy achieved: 83,7%.

Table 4.9: Class level precision recall values for model 11 evaluated on entire personal dataset (adjusted to 12 FPS).

Class	Precision	Recall
Swiping Right	0.94	0.30
No gesture	0.68	0.98
Swiping Down	1.00	0.80
Swiping Left	0.98	0.99
Swiping Up	1.00	0.96

4.2.3.3 Fine tuning on personal dataset: 12 FPS

Fine tuning was performed using model 11, after which evaluation was performed on the previously unseen 20% data. The result of this evaluation can be observed in the form of a confusion matrix in Figure 4.6, as well as precision recall values in Table 4.10. The accuracy of this evaluation was estimated to 97.5%.

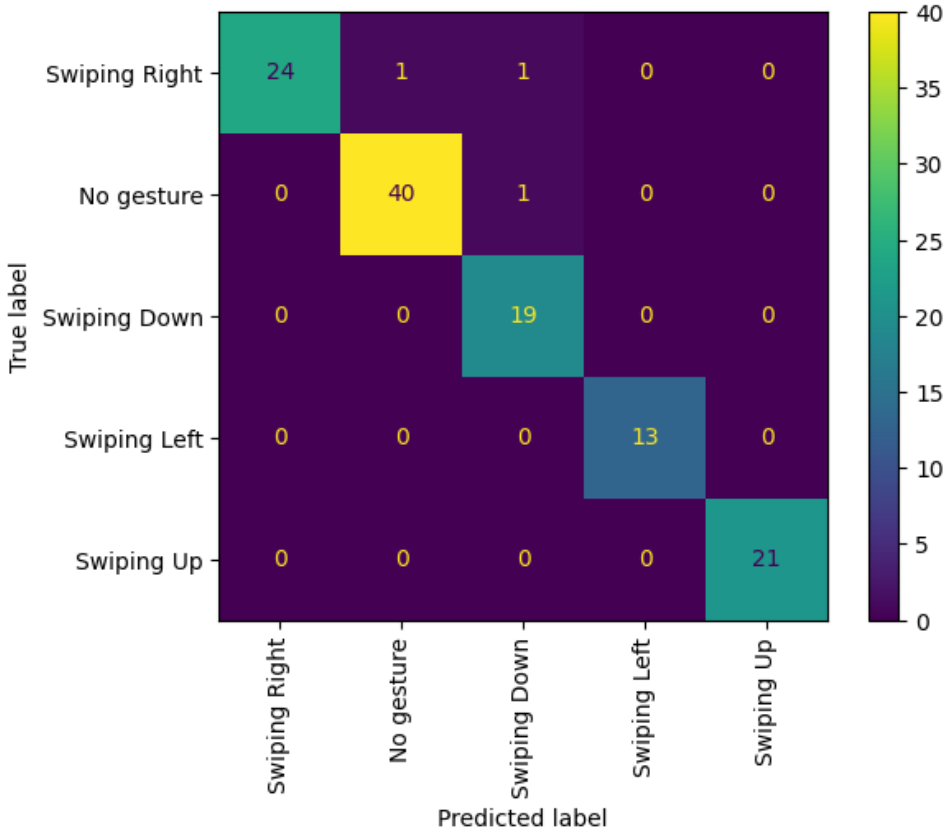

Figure 4.6: Confusion matrix from using model 11 after fine tuning on test part of personal dataset. Note that the dataset was converted to 12 FPS for this evaluation. Accuracy achieved: 97,5%.

Table 4.10: Class level precision recall values for model 11 after fine tuning evaluated on test part of personal dataset

Class	Precision	Recall
Swiping Right	1.00	0.92
No gesture	0.98	0.98
Swiping Down	0.90	1.00
Swiping Left	1.00	1.00
Swiping Up	1.00	1.00

4.3 Integrated system

A real-time application was created with the functionality to recognize both static and dynamic hand gestures, where the result is shown as a live video feed, with the prediction and hand landmarks drawn on top of the video. Latency measurements for this app, including its subsystems were performed as described in section 3.4.2. Results in form of a bar chart and a pie chart are shown in Figures 4.7 and 4.8.

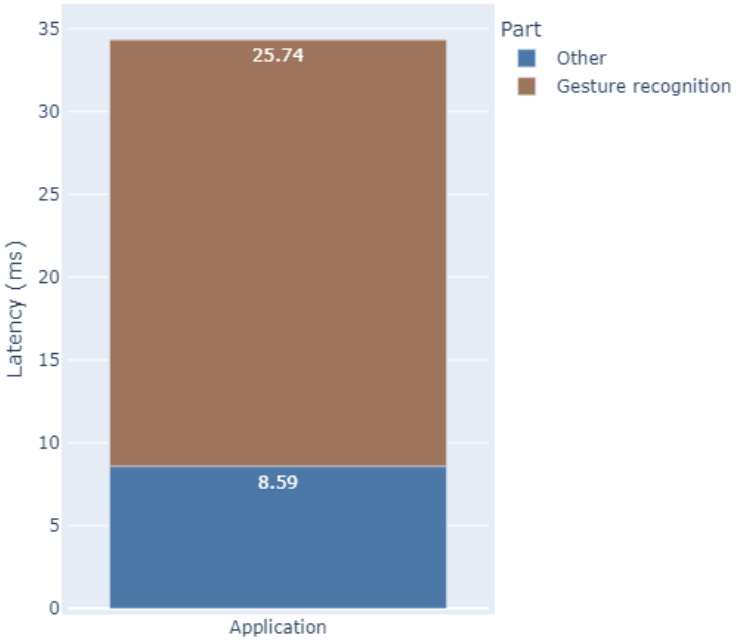

Figure 4.7: Overview of latency in application per captured frame. Note "other" includes application specific processes such as displaying a live video, drawing hand landmarks and displaying text of the predictions from the classifiers.

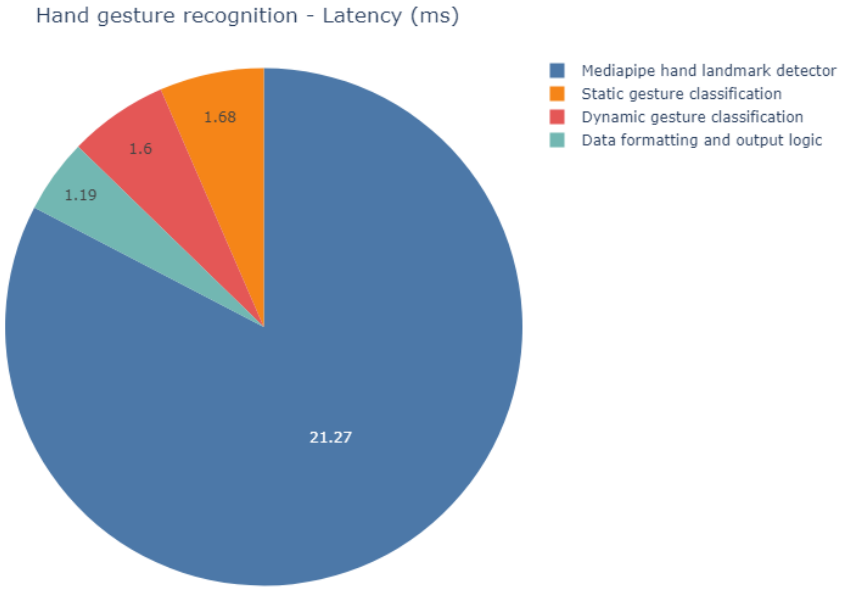

Figure 4.8: Individual contributions of the subsystems within the hand gesture recognition model.

5
Discussion

5.1 Static hand gesture classification on Hagrid & Jester dataset

The result of the performance comparison of the models developed, can be seen in Table 4.1, where the best performing model 1, only very marginally outscores the other neural networks including the same model trained on an augmented version of the dataset. The accuracy of this feed forward neural network was estimated to 93,2% on the test dataset, while still achieving relatively high recall for all classes (see Table 4.2). In fact, the lowest recall value was for the fist class with a value of 0.89, which implies 89% of all fist images were correctly identified as fist. However it is interesting to note that the precision value for the no gesture class is comparatively much lower than the others, suggesting many false predictions were made by predicting the no gesture class. This is also evident by inspecting the no gesture column in the confusion matrix in Figure 4.1, where it seems clear that almost all errors made were false predictions of the no gesture class. As performance of the model is very tightly coupled to the feature extraction process, a root cause analysis was performed related to MediaPipe's feature extraction algorithm's ability to detect hands in the images. As shown in Table 4.3 a very large majority (76.3-88.7%) of all errors on the positive classes occur when the MediaPipe algorithm has completely failed to detect a hand in the image. In fact, 100% of the times MediaPipe failed to detect a hand in an image of a positive class, an false prediction was made. However considering that the no gesture class does not require any hands to necessarily be present in the image, defaulting predictions to this class when no hands can be detected makes a certain amount of sense. Unfortunately, with this analysis, improving the model further also becomes very difficult, since so many of the errors are due to poor feature extraction.

5.2 Dynamic hand gesture recognition on Jester dataset

The best performing model for dynamic hand gesture recognition was model 11, as can be shown by inspecting Tables 4.4 and 4.5. Unlike, the case with the static hand gesture recognition models, training on an augmented version of the dataset resulted in higher accuracy, if only marginally so. The accuracy of model 11 was estimated to 88.1% on the test part of the Jester dataset. While this number is pretty high

it is also not quite as high as some state of the art algorithms, with results ranging from 84.75-96.72% accuracy for different models (except evaluated on the full Jester dataset and not just a subset) [23]. By inspecting Table 4.6, it can be observed that the precision of the no gesture class is comparatively much lower than the other classes, suggesting many errors are due to falsely predicting the no gesture class. Additionally, the confusion matrix looks very similar compared to that of the static hand gesture recognition model in Figure 4.1, where most of the errors are located in the no gesture column. Despite these similarities it is not trivial to assume that the MediaPipe feature extraction algorithm has as a significant impact on limiting the model's performance, since each classification is made based on a sequence of images instead of a single image. As such failing to detect a hand in one image of a sequence, may not necessarily be as detrimental, since the other images may be correlated and effectively allow the model to infer the missing information. However using Table 4.7 it can be observed that the average number of hands when falsely predicting no gesture is indeed comparatively much lower when correctly predicting a positive class. While this is not conclusive evidence that false negatives are the root cause for many of the model's errors, it does seem very plausible since the model operates using only a small fraction of the images it typically does when performing a correct classification.

Going further in the analysis of errors that can be attributed to the MediaPipe feature extraction algorithm it can also be shown that false positives might have a significant negative impact on model performance as well. As shown in section 4.2.2.2 using Bayes theorem, the risk of a false classification by the model show a significant increase (from 12.2% to 19.7%) if a false positive (i.e. detecting a hand where none exist) was made by the MediaPipe feature extraction algorithm. Additionally, because of their relatively frequent occurrence in the dataset, they could potentially account for 28.9% of all the errors, if one can assume a cause effect relationship. While a cause effect relationship has not been conclusively proven in this report, it is not completely unreasonable to think one might still exist. Especially given the previous analysis where it was shown that only a relatively small fraction of the images included a hand detected by the MediaPipe algorithm when a false prediction of the no gesture class was made. With few images to base the classification on, and the risk that one or more of those images are a false positive (i.e. a hand detected where none exist), an increased risk for miss classifications does not seem entirely illogical.

5.3 Dynamic hand gesture recognition on personal dataset

Finally, with regards to how well the model generalizes to the real world, evaluating performance on the personal dataset (30 FPS) model 11 achieves 78.0% accuracy. Interestingly, when evaluated on the same dataset except removing images to adjust it to 12 FPS, it scored significantly higher (83.7%). While this may seem counter intuitive considering previous arguments where lower fraction of images with hands detected resulted in higher risk of errors, this is not quite the same case. While

the total number of images are indeed greater, and thus increase the likelihood that the MediaPipe algorithm can detect more images with hands in them, the fraction of images with hands in them may still be unchanged. Additionally, because of the shorter time interval between consecutive frames, it is also possible that the temporal patterns the model has been trained to capture become different. After all, by increasing the capture rate, the distance the hand moves between frames is going to be smaller (assuming the hand gesture was performed the same way). Hence for these reasons, lowering the model's frame rate by removing images, might be justified. One similarity between the evaluation on the Jester dataset and this dataset, is how the errors in the confusion matrices (see Figure 4.4 and 4.5) are almost exclusively located in the no gesture column, suggesting the root cause for this is not specific to a single dataset. One interesting difference between the evaluations on the two different datasets and the Jester dataset, is that the recall values for the swiping right class is much lower than that measured on the Jester dataset (see Tables 4.8, 4.9, 4.6). In fact the recall value 0.3 observed in Table 4.9 implies only 30% of the swiping right gestures are correctly identified as swiping right. Given, the comparatively high values for the other classes this is very interesting, and potentially related to specific user behaviour when collecting that gesture. However, as can be observed after fine tuning model 11 on 80% of the dataset, and evaluating it on the remaining 20% the recall value is much improved (see Table 4.10). In fact the overall accuracy for this fine tuned model is 97.5%, with almost no errors left in the no gesture column (see Figure 4.6). While the exact root cause for why the performance improved so drastically is not known, one possible explanation could be that it fine tuned the model to recognize the hand gestures using fewer images, which became possible due to how uniformly the gestures were performed since they were made by a single person as opposed to 1300 in the Jester dataset. If this is indeed the case, then applying the model to different users, could prove in worse overall performance. Regardless, since fine tuning the model could be performed so effectively with relatively little data, it could open up possibilities where fine tuned models are performed for every user by prompting them to collect a few examples of every gesture and tune a base model in the background.

5.4 Latency measurements on demo application

As can be shown in Figure 4.7, the application including the hand gesture recognition model has a total latency below 35 ms, suggesting it works well for real-time applications (since 35 ms corresponds to approximately 28 frames per second). Unsurprisingly, most of the latency can be attributed to the gesture recognition model (25.74 ms). However it is interesting to note that most of this latency is attributed solely to the feature extraction process by the MediaPipe hand landmark detector. While this makes a certain amount of sense, since the feature extraction model needs to process more data because of its much larger number of inputs than the classifiers, it is also clear that there is a lot to gain by optimizing this step. One potentially very significant improvement one could try would be to utilize the MediaPipe hand landmark detector implementation in C++ instead of Python, as C++ typically enables running code much faster. Comparably improvements such as configuring

the code to enable running the static and dynamic hand gesture classifiers in parallel would only yield marginal improvement at best.

5.5 Ethical analysis

AI systems today are increasingly used for high-impact tasks such as within healthcare, law enforcement and education [25]. As a result, reports of errors and failures have also increased [25]. Despite the potentially serious risks of these errors, there have previously been relatively little information available to people interested in using trained machine learning models, thereby making the process of understanding the risks even harder [25]. However, as of February 2024, the European Union passed the EU AI Act, governing which AI models can lawfully be developed with the aim of minimizing harm to health and safety of people and their fundamental rights [26]. Thus, it is now more important than ever to provide clear, detailed information about the model's intended use case, and its known limitations.
In this report, a system for recognizing both static and dynamic hand gestures have been developed. However, as analysis of it as a complete system using real world data was only performed with one person (due to time constraints), much of its limitations are still unknown. Additionally, for the same reasons, analyzing the model for biases with regards to properties such as skin tone, sex and age were also not been possible. However it should be mentioned, that during development of the system care was taken to minimize these risks such as by strategically selecting the datasets used to train the classifiers. For instance, Hagrid was selected for its large size and diversity in terms of age and gender (37 583 unique individuals) [27]. Similarly Jester was selected because of its large size (1300 unique individuals) [22], thus reducing the risk of introducing bias by selecting a small number of people, which may not reflect the true population. Furthermore, the choice of feature extraction algorithm was also analyzed before it was selected. Fortunately, the MediaPipe hand landmark detector proved to have extensive information available about it in terms of its training processes and the types of analysis performed to check the model for biases in terms of skin tone and sex [28]. Since the model passed those tests, and met the target use (i.e. hand gesture recognition for research purposes) this was likely a good choice [28]. Despite this, and as previously mentioned, much of the performance of this model still remains unknown when it comes to real life performance and as such is only recommended to be used for research purposes.

6
Conclusion

A real time application capable of recognizing both static and dynamic hand gestures were successfully developed. The system, based on the MediaPipe hand landmark detector for feature extraction and two separate classifiers for static and dynamic hand gestures, proved fast and relatively accurate. Static hand gesture recognition on a subset of a combined Hagrid and Jester dataset was measured to 93.2% accuracy, whereas dynamic hand gesture recognition on a subset of Jester dataset was measured to 88.1%.

Analysis of the errors performed on both classifiers, resulted in strong indications that the MediaPipe feature extraction algorithm was limiting performance greatly by either failing to detect hands where a hand exist, or detecting hands were none existed. Hence one recommendation for future work is to improve or fine tune the MediaPipe hand landmark algorithm in order to achieve even higher accuracy.

Without fine tuning the feature extraction algorithm itself, but the dynamic hand gesture classifier, accuracy could be improved from approximately 83.7% to 97.5% using only a small scale dataset of one person. This model has however not been thoroughly tested on real life data and is thus only recommended to use for research purposes.

Bibliography

[1] Jimin Yu, Maowei Qin, and Shangbo Zhou. Dynamic gesture recognition based on 2d convolutional neural network and feature fusion. *Scientific Reports*, 12, 2022.

[2] Google AI for Developers. Hand landmarks detection guide. `https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker#task_details`, 2024. [Accessed 11-10-2024].

[3] Fernando Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014–.

[4] Xuedong Zhu, Mingxing Lin, Yubo Liu, Wenjing Fan, and Chaoguo Shi. Human-computer interaction technology for upper limb rehabilitation based on gesture recognition control. In *2022 12th International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 1072–1076, 2022.

[5] Amal Hameed Khaleel, Thekra HayderAli Abbas, and Abdul-Wahab Sami Ibrahim. Enhancing human-computer interaction: A comprehensive analysis of assistive virtual keyboard technologies. *Ingénierie des Systèmes d'Information*, 28(6):1709–1717, 2023.

[6] Danhua Huang and Shuaiqiu Xiang. Speech recognition and intelligent translation under multimodal human–computer interaction system. *Journal of Intelligent Systems*, 33(1):20230192, 2024.

[7] Reena Tripathi and Bindu Verma. Skeleton data is all about: Dynamic hand gesture recognition. *2023 Seventh International Conference on Image Information Processing (ICIIP)*, pages 576–585, 2023.

[8] Bogdan Ionescu, Didier Coquin, Patrick Lambert, and Buzuloiu Vasile. Dynamic hand gesture recognition using the skeleton of the hand. *EURASIP Journal on Advances in Signal Processing*, 2005, 08 2005.

[9] Reena Tripathi and Bindu Verma. Skeleton data is all about: Dynamic hand gesture recognition. In *2023 Seventh International Conference on Image Information Processing (ICIIP)*, pages 576–585, 2023.

[10] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking. *ArXiv*, abs/2006.10214, 2020.

[11] Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *ArXiv*, abs/1402.1128, 2014.

[12] Shravya Bhaskara, Dadi Jaideep, Ganta Sai Sampath, Arya Arun, and Prasanth M. Warrier. Dynamic hand gesture recognition for video playback control.

In *2023 IEEE Fifth International Conference on Advances in Electronics, Computers and Communications (ICAECC)*, pages 1–6, 2023.

[13] Vaidehi Sharma, Abhishek Sharma, and Sandeep Saini. Real-time attention-based embedded lstm for dynamic sign language recognition on edge devices. *Journal of Real-Time Image Processing*, 21(2):53, Mar 2024.

[14] Christopher Olah. Understanding lstm. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015. [Accessed 12-10-2024].

[15] Gursewak Singh. Demystifying lstm weights and bias dimensions. `https://medium.com/analytics-vidhya/demystifying-lstm-weights-and-biases-dimensions-c47dbd39b30a`, 2020. [Accessed 15-10-2024].

[16] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *ArXiv*, abs/1804.07612, 2018.

[17] pytorch documentation. `https://pytorch.org/docs/stable/generated/torch.nn.utils.rnn.pack_padded_sequence.html`, 2023. [Accessed 13-10-2024].

[18] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, 2017.

[19] Francesco Periti, Haim Dubossarsky, and Nina Tahmasebi. (chat)gpt v bert: Dawn of justice for semantic change detection. *ArXiv*, abs/2401.14040, 2024.

[20] Srijanie Dey. Deep dive into self-attention by hand. `https://towardsdatascience.com/deep-dive-into-self-attention-by-hand-%EF%B8%8E-f02876e49857`, 2024. [Accessed 13-10-2024].

[21] Kai Ming Ting. *Precision and Recall*, pages 781–781. Springer US, Boston, MA, 2010.

[22] Qualcomm AI Research. Jester dataset. `https://www.qualcomm.com/developer/software/jester-dataset`. [Accessed 13-10-2024].

[23] Yun Zhang and Fengping Wang. Handformer: A dynamic hand gesture recognition method based on attention mechanism. *Applied Sciences*, 13(7), 2023.

[24] Zhengzheng Xing, Jian Pei, and Eamonn J. Keogh. A brief survey on sequence classification. *SIGKDD Explor.*, 12:40–48, 2010.

[25] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 220–229, 2019.

[26] The pre-final text of the eu's ai act leaked online. `https://www.whitecase.com/insight-alert/pre-final-text-eus-ai-act-leaked-online`, 2024. [Accessed 16-10-2024].

[27] Alexander Kapitanov, Karina Kvanchiani, Alexander Nagaev, Roman Kraynov, and Andrei Makhliarchuk. Hagrid–hand gesture recognition image dataset. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 4572–4581, 2024.

[28] MediaPipe. Model card hand tracking (lite/full) with fairness oct 2021. `https://storage.googleapis.com/mediapipe-assets/Model%20Card%20Hand%`

20Tracking%20(Lite_Full)%20with%20Fairness%20Oct%202021.pdf, 2021. [Accessed 16-10-2024].

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY