

Hand Gesture Classification using Millimeter Wave Pulsed Radar

Eda Daganan

January 10, 2020

Abstract

Millimeter wave pulsed radar has found many applications, among them hand gesture sensing, which this work has as purpose. This application has already shown good potential, [1], and here in this work robustness aspects are taken into account. A classification task was defined, where distinction should be made between five different small hand gestures, recorded by 18 people in total. The classification was performed using 1D and 2D convolutional neural networks, that were compared to a k-Nearest Neighbour classifier as a benchmark. Two different approaches of feature extraction for the classifiers were attempted. On one hand, a spectrogram feature, holding time-frequency information, that showed very good performance; on the other hand an estimation of the sparse room impulse response using Elastic net and Wideband integrated dictionaries, with less satisfactory results. Special attention is given to the analysis of how well a classifier can perform on many people, even if their data is completely excluded from the training set.

Acknowledgements

This work would not have been accomplished without the involvement of several people. I want to thank my both supervisors, Prof. Andreas Jakobsson and Dr. Peter Almers, for offering me the opportunity and for incessantly engaging yourselves and supporting me in my work. Further thanks to all colleagues at Acconeer for great help and collaboration, and to Ola Björnsson for guiding me in the use of the computational resources at LUNARC. Finally, thanks to all the people who helped me record data.

Contents

1	Introduction	4
1.1	Motivation and Previous Work	4
1.2	Objectives and Problems Formulation	4
2	Radar System Overview	5
2.1	Raw Data	5
2.2	Moving Objects	7
2.3	IQ Demodulation	8
2.4	Sparse Mode	9
3	Data acquisition	11
3.1	Measurement Setup	11
3.2	Measurement Settings	12
3.3	Dataset 1 and 2	12
3.4	Dataset 3	12
4	Feature Extraction	14
4.1	Spectrogram Feature	14
4.1.1	Hyperparameters	17
4.2	Impulse Response Feature	19
4.2.1	Elastic net	19
4.2.2	A sparser solution by wideband integrated dictionaries	20
5	Classification Algorithms and Model Selection	21
5.1	k-Nearest Neighbour	21
5.2	Convolutional Neural Networks	21
5.3	Complexity Analysis	23
5.3.1	Feature Preprocessing	24
5.3.2	KNN	24
5.3.3	CNN	24
5.4	Cross-Validation	25
6	Results	27
6.1	Feature Extraction	27
6.2	Classification Results on Dataset 1 and 2	32
6.3	Classification Results on Dataset 3	33
6.3.1	Test Results 5-fold Cross-Validation	33
6.3.2	Test Results Leave-One-Out Cross-Validation	33
6.4	Complexity Analysis	36
7	Discussion	38
7.1	Problems with the Setup	39
8	Conclusions and Future Work	41

1 Introduction

Radar is an acronym for radio detection and ranging, and it is an instrument that can detect radio frequency signals scattered from distant objects [2]. By emitting electromagnetic waves from a transmitter antenna, of a carrier frequency (CF), the signal propagates in space, reflecting on objects in its path. The backscattering that returns to the sensor is detected by a receiver antenna and is further processed in the system. The technology in essence therefore provides a "measurement of distance, inferred from a relative time-delay and an assumed or known speed of travel of radio wave" [3].

Radar technology dates back to the first experiments in the late 19th century, to a leap in development during World War II, [4], primarily being developed for military use and for large-scale distance measurements. A more compact radar system, operating on GHz scale has been present since 1990s [5]. As [1] points out, the modern radar technology has advantages, necessary for today's applications: "The RF spectrum has several highly attractive properties as a sensing modality for interactive systems and applications: the sensors do not depend on lighting, are extremely fast and highly precise; and can work through materials, which allows them to be easily embedded into devices and environments". Recent applications have been observed in, e.g., measuring heart rate and respiratory signals, [6] and tumour detection, [7]. All applications can be summarized by three primary goals: detection, localization and classification [8].

1.1 Motivation and Previous Work

When it comes to gesture sensing – the ability to recognize certain predefined hand movements from the backscattered reflections – using pulsed millimeter wave radar systems, the topic has recently attracted notable attention, both in academia and in industry. Much research and work has been done by Project Soli [1], which has also led to the integration of a radar sensor permitting gesture control into Google's smart phone, Pixel 4, [9]. Research on the same technology has also been carried out at the Department of Electrical and Informational Technology at Lund University, see for example [10], [11], where a pulsed radar system has been designed and further employed to record and delineate hand gestures, applying an IQ demodulation to the data. Gesture classification has also been done by [12], where exactly the same Acconeer hardware system as in this work was used. There, hand gestures were classified using convolutional and recurrent neural networks, with good results. As an extension to that project, this work treats further options of generating the radar data and extracting features from it.

1.2 Objectives and Problems Formulation

This project is aimed at developing ways of processing radar data from Acconeer's pulsed radar sensors to obtain distinguishable hand gesture class features. To do so, the main areas of investigation were defined to be: comparing different types of radar data, ways to process them and algorithms to classify them. In alignment of what Google's project Soli has been focusing on, the hand gestures should be small, *micromotions*, taking no real advantage of changes in location. To narrow the problem formulation and better visualize an application, the set of gestures considered was chosen to be designed as if they were to be used in a small electrical device, such as e.g. a smartphone or a tablet. The general approach to the problem was to aim for robustness in a practical scenario.

2 Radar System Overview

Many forms of pulsed radar systems are designed to emit short pulses of high power and internally matching the reflected signal, [2], which is described in the following section. The matched result can then be further processed; a brief description of IQ demodulation will be presented.

2.1 Raw Data

Let the emitted radar pulse be modeled as

$$x(t) = A(t) \cos(2\pi f_0 t), \quad (1)$$

where $A(t) = e^{-t^2/2\lambda^2}$ is a Gaussian envelope and $f_0 = 60$ GHz. An illustration of the pulse is shown in Figure 1. The parameter λ in the envelope function $A(t)$ is related to the width of the pulse.

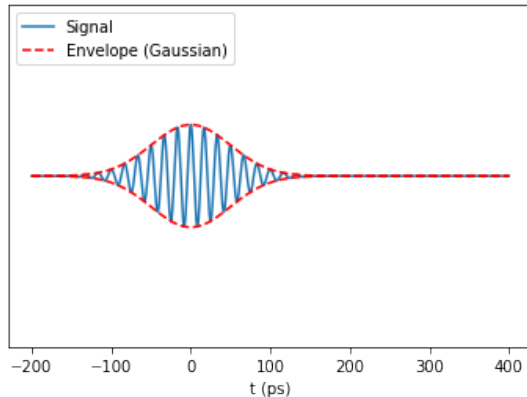


Figure 1: The signal $x(t)$ sent from the transmitter antenna.

The transmitted signal then propagates in space, and reflects on objects in its path. Assume that there is a single object located at a distance d from the radar. Reflections from this object backscatter, with parts of the signal returning to the radar sensor with some changes:

- a delay depending on d
- a loss in energy (change in envelope),

which can be expressed as

$$x_{\text{propagated}}(t) = \tilde{A}(t - \tau) \cos(2\pi f_0(t - \tau)).$$

The delay of the propagated pulse gives most surely rise to a phase shift as compared to the emitted pulse. The loss in energy depends on the reflective properties of the object and the distance to the sensor. In general, the principle of transmitting pulses of maximum peak power and short pulse duration results in low average power, which has the consequence of a limited maximum detection range [2]. As an example, the energy in a signal that has reflected on a spherical object is proportional to $1/d^4$. Figure 2 illustrates a simulated transmitter signal and its reflection when it returns to the sensor. The virtual object in the simulated example is at $d = 10$ cm distance. Note that there is a direct connection between the time delay variable, t ,

and the distance to the object, d , since the wave travels $2d$ before returning, in our example $d = 10$ cm implies that the pulse returns after $2d/c = 2 \cdot 10^{-1}/3 \cdot 10^8 \approx 667 \cdot 10^{-12}$ s. This time shift is referred to as the *time of flight*.

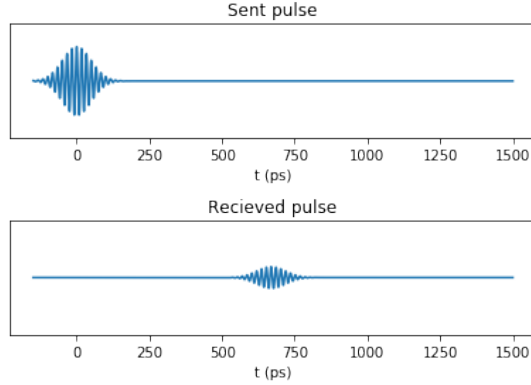


Figure 2: Transmitted signal and its reflected signal on an object at 10 cm distance.

The system reconstructs the signal by an analogue convolution. The approach is as follows: internal signals of different delays are generated, and these are matched with the reflected signal. The internal signals thus have the form:

$$x_{\text{internal}}(t - \tau_i) = A(t - \tau_i) \cos(2\pi f_0(t - \tau_i)).$$

For each delay, the internal receiver pulse x_{internal} is mixed with the reflected signal (that is, multiplied element by element). Figure 3 illustrates this for 3 lags. In the middle panels of 3a-3c, is the received pulse, after having backscattered at an object at 10 cm distance. In the top panels, are the internal pulses, delayed of increasing lags. In the bottom figures are the results of mixing the received pulse with the internal, delayed.

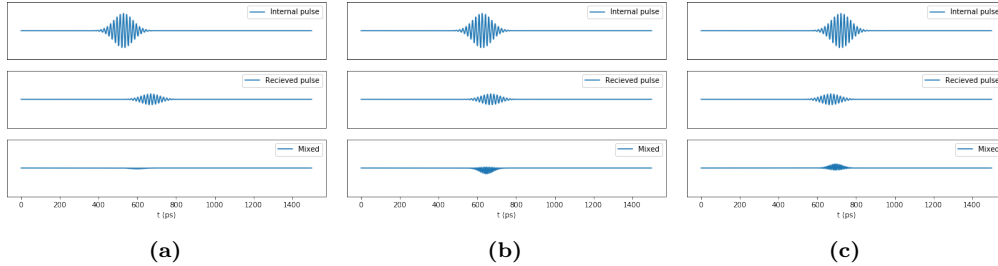


Figure 3: Mixing of internal pulses and the propagated signal.

If the reflected pulse is denoted by $x_{\text{propagated}}(t)$, the mixed signal (or elementwise product) is exactly

$$x_{\text{internal}}(t - \tau_i)x_{\text{propagated}}(t) = x_{\text{internal}}(\tau_i - t)x_{\text{propagated}}(t),$$

for all delays τ_i , the equality holding by symmetry. By integrating (in practice summing) over t , one obtains the convolution

$$y(\tau_i) = x_{\text{internal}}(\tau_i) * x_{\text{propagated}}(\tau_i) = \int_t x_{\text{internal}}(\tau_i - t)x_{\text{propagated}}(t)dt. \quad (2)$$

Each delay τ_i therefore results in an estimation of a point of the convolution $y(\tau) = (x_{\text{internal}} * x_{\text{propagated}})(\tau)$. The mixing is repeated for a fixed set of delays (each time with a new transmitted pulse), yielding a sampled version of the full convolution. The resulting vector is referred to as the *raw data*. The difference in lag of the delays are as small as to fit several samples per wavelength. The delays are also determined by the user, if he or she has a vague idea of how delayed $x_{\text{propagated}}$ is and thus limits the search range. Figure 4 illustrates the resulting sample convolution (delaying, mixing, summing) of the simulated example, together with the convolution in (2), obtained by Python numpy’s `convolve`.

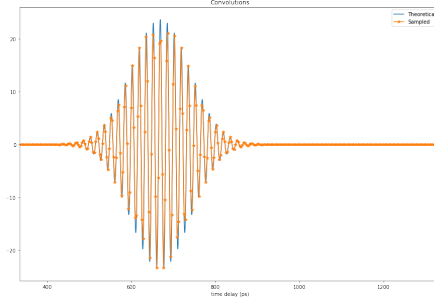


Figure 4: A simulation of the system’s analogue convolution (the raw data) and the theoretical convolution.

2.2 Moving Objects

Assume there is a moving object in front of the sensor with constant velocity v . See the upper illustration in Figure 5. If the system samples with sweep frequency f_s , so that the time between two sweeps is $\Delta t_s = 1/f_s$, then the object will have moved $v\Delta t_s$ each sample. The wave will then have travelled twice this distance, $2v\Delta t_s$, or equivalently $2v\Delta t_s/\lambda$ wavelengths. The returning pulse has thus a change in phase of

$$\Delta\phi = 2\pi \frac{2v\Delta t_s}{\lambda} = \frac{4\pi v\Delta t_s}{\lambda}.$$

See Figure 6 for an illustration of this. The phase shift is highly informative, since it is in direct relation to the velocity, and extracting it is crucial to well capture hand movements.

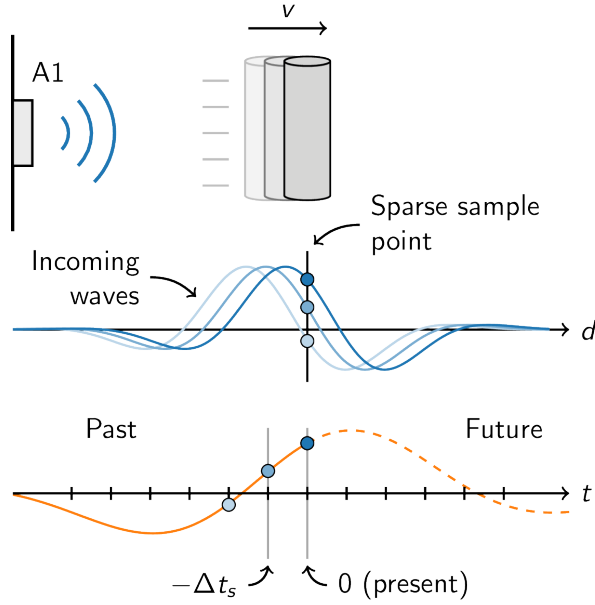


Figure 5: Images illustrating the phase shifts as effect of moving objects, taken from [13].

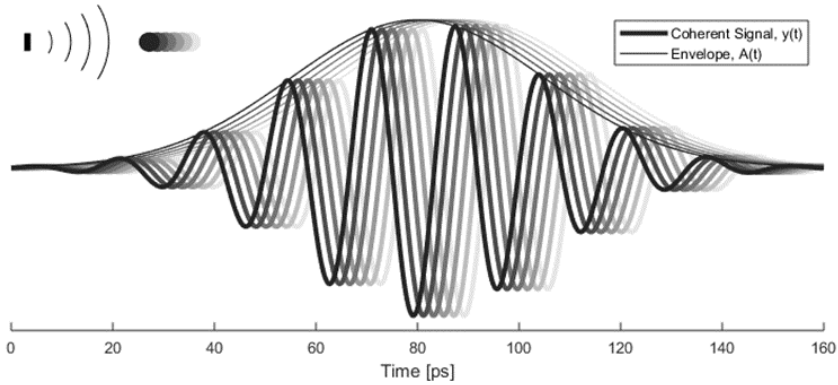


Figure 6: Images illustrating the phase shifts as effect of moving objects, taken from [14].

2.3 IQ Demodulation

One possible processing of the raw data is the IQ demodulation, see e.g. [15] for a thorough introduction. This processing offers a possibility to capture the phase shift of the measured signal compared to the transmitted pulse.

By mixing the reflected signal $A(t - \tau) \cos(2\pi f_0 t + \Delta\phi)$ by the complex signal $e^{-i2\pi f_d t}$, with demodulation frequency f_d , the result becomes

$$A(t - \tau) \cos(2\pi f_0 t + \Delta\phi) e^{-i2\pi f_d t} = \frac{A(t - \tau)}{2} (e^{i2\pi(f_0 - f_d)t + i\Delta\phi} + e^{-i2\pi(f_0 + f_d)t + i\Delta\phi}),$$

by Euler’s formula. If f_d is chosen close to f_0 , the first term in this sum will have frequency close to 0 and the second approximately the double frequency. A low pass filter will therefore isolate the first term and return

$$\frac{A(t - \tau)}{2} e^{i\Delta\phi}. \quad (3)$$

The result is a complex signal with the argument being the phase shift $\Delta\phi$ and the modulus of the envelope $A(t - \tau)/2$.

This demodulation is applied to the raw data, from which one obtains the *IQ signal*, which is here denoted by

$$y_{IQ}(t_s, d), \quad t_s = 0, \dots, N_{t_s} - 1, \quad d \in \{d_0, \dots, d_{N_d-1}\},$$

where t_s denotes the sweep index, that is, the index of the analogue convolution originating from each emitted pulse $x_{\text{propagated}} * x_{\text{internal}}$; d denotes the distance to the sensor, which is in direct relation to the delay variable within the sweep.

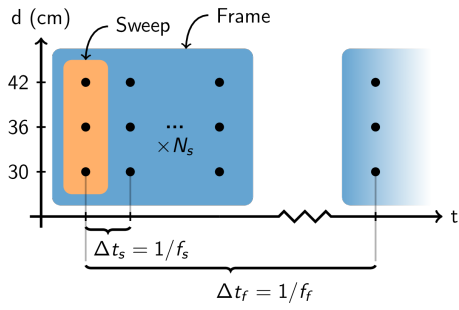
2.4 Sparse Mode

The raw data described up to this point is generated by very small steps in pulse delay, with the aim of high spatial resolution. There is another version of generating raw data – which turns out to be a strong candidate for describing motions – but on the contrary is using a very coarse sampling of the pulse delay. If the IQ raw data is delayed as to fit several samples per wavelength (effectively approx. 2.5 mm in space), the sparse sampling mode delays the samples for multiples of 6 cm. The advantage of this is that it can be done at a much higher frequency and thus track movements with higher temporal resolution. In Figure 5 a schematic figure describes the procedure. Formally, denote the sparse signal by

$$y_{\text{sparse}}(t_f, t_s, d), \quad t_f = 0, \dots, N_{t_f}, \quad t_s = 0, \dots, N_{t_s} - 1, \quad d = 6k, k \in \mathbb{Z}^+, \quad (4)$$

where t_s denotes the sweep index, t_f the *frame* – the sequence of sweeps – and d the distance to the sensor. Figure 7a illustrates an example of the sampling scheme. It is in the nature of how the sparse signal is generated that a burst of consecutive sweeps are generated before a pause, such that the frames t_f are separated. The frames are generated according to a frame update rate, whereas the sweeps within a frame are repeated according to another sweep update rate.

This kind of raw data therefore gives a spatially sparser representation but, thanks to a high sweep rate, a temporally more dense image of the target scene. What is more interesting is therefore how the reflections $y_{\text{sparse}}(t_f, t_s, d)$ are evolved over time, t_s . See Figure 7 for an example of this representation.



(a) Illustration of the sparse sampling scheme. For each distance d , N_{t_s} sweeps are repeated with frequency f_s , into a *frame*. The frames, in turn, are repeated with frequency f_f . The picture is taken from [16].

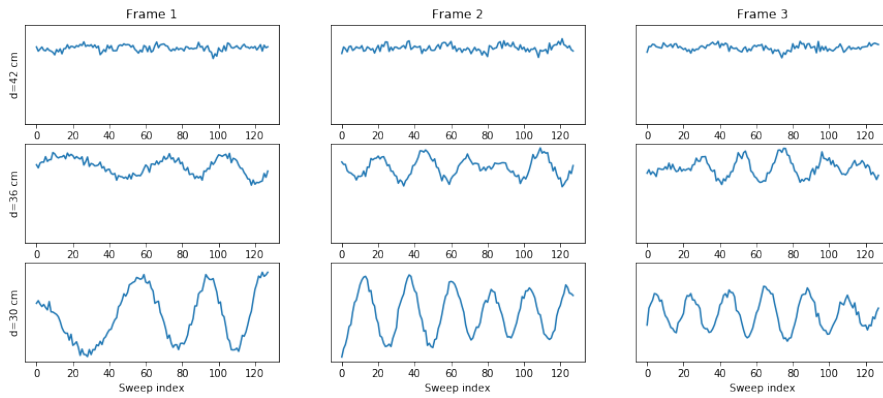


Figure 7: Example of sparse data ($N_{t_s} = 128$).

3 Data acquisition

3.1 Measurement Setup

For the data collection, two XR112 sensors, connected to a XC112 connector board, mounted on a Raspberry Pi were used. They were installed behind the glass of a mockup mobile phone, which was placed on a table. The gestures were performed 5 – 30 cm above the two sensors, in an area which will be denoted *target scene*.

Five gestures constituted classes in the data sets, see also Figure 8:

- Palm hold (static)
- Swipe
- Button press (between index finger and thumb, twice)
- Checkbox (index finger pointing, moving towards the sensor)
- Erase (flat hand moving as if erasing something)

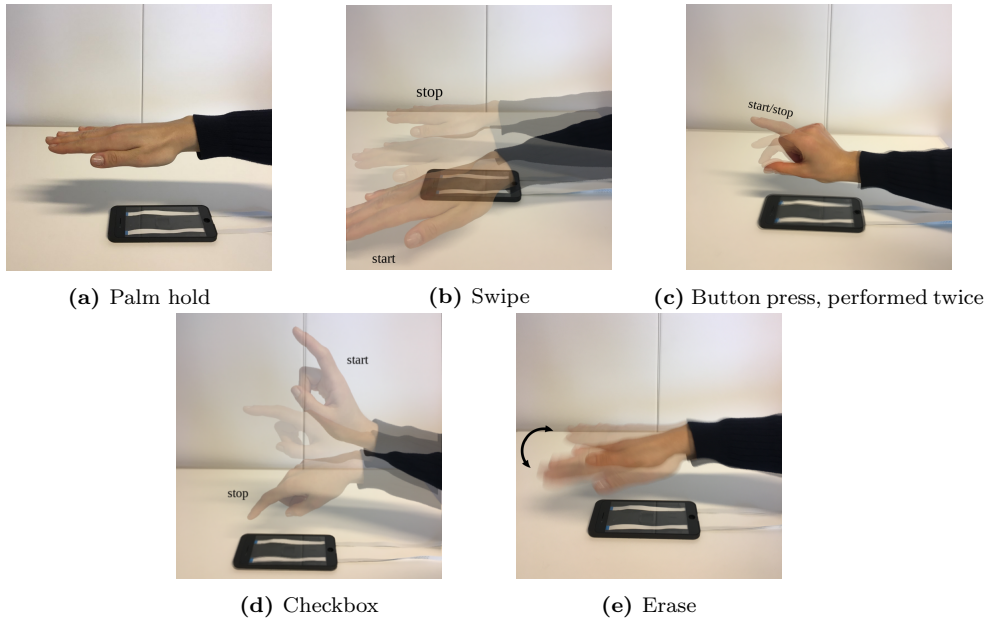


Figure 8: The gesture set.

The Button press gesture is one that was introduced by Google Soli [1]. To get clean data of only the relevant movements, it was decided that for the classes Palm, Button, Checkbox, and Erase, the gesture starts when the hand is placed above the sensor. To mark the start of the gesture, a button was pressed down throughout the full gesture. That was enough to give information of the gesture start, the gesture stop and the length, which was analyzed at an early stage of the project to choose gesture duration, namely 1.1 s. That time corresponds to a time window of 186 sweeps (IQ) and 20 frames (sparse). When gathering the data sets, the start of the gesture was indicated by pressing a button, to be able to extract the relevant parts of the data.

3.2 Measurement Settings

The following radar settings were used when recording the data:

Table 1: Settings for Acconcers XR112 sensors. The parameters Frame rate and Sweeps per frame are unique for sparse mode.

Dataset	Service	Search range (cm)	Sampling mode	Pulse length (ps)	Step-size	Gain	Sweep rate (Hz)	Frame rate (Hz)	Sweeps per frame
1,3	Sparse	[18, 36]	A	800	1	0.5	2500	18	128
2	IQ	[13, 35]	B	50	2	0.8	175		

3.3 Dataset 1 and 2

Initially, two data sets were gathered from one person: one constituting of IQ demodulated data, one constituting of the sparse data. Data was collected by repeating one gesture a fixed number of times, a *session*, (roughly) according to the following protocol:

- For each gesture:
 - Repeat 20 times:
 - * Perform the gesture
 - * Retract the hand
 - Pause

One session was thus performed for each class at a time, with a small pause in between two sessions. Between each repetition within one session, the people performing gestures were instructed to retract the hand of the target scene, to introduce more variability. Moreover, the data gathered from both sensors during one repetition of a gesture were used, resulting in a data set of approximately 6500 examples, see Table 2.

3.4 Dataset 3

During the second iteration of the project, data was gathered from other people performing gestures. They were instructed how to do the gestures and got to practice the gestures a couple of times, until they claimed to be ready. Most of the people were told when to start the gestures, as someone else was marking the start of the gesture. One person used the left hand; the others the right. Data was recorded according to

- For each gesture:
 - Repeat 10 times:
 - * Perform the gesture
 - * Retract the hand
 - Pause

Table 2: The distribution of classes within the datasets.

Dataset	People	Datatype	Palm hold	Swipe	Button press	Checkbox	Erase	Total
1	1	Sparse	1272	1338	1336	1336	1282	6564
2	1	IQ	1360	1370	1374	1380	1244	6728
3	18	Sparse	1340	1366	1344	1340	1342	6732

4 Feature Extraction

The signature features, or the data representation, should be done so that different classes can be distinguished as well as possible, but the feature selection then naturally also depends on the gestures. The selection of features and gestures were therefore performed quite in parallel. An initial set of possible gestures was proposed, with practical user applications in mind. The data corresponding to these were then investigated (also for suitable choices of parameters of the data generation), as was possible features. At this point, five classes were chosen by visual inspection, which are the ones presented in the report.

4.1 Spectrogram Feature

The radar sensor has a limited resolution and could for example not distinguish very well between different parts of the hand. Therefore, instead of creating a feature delineating the spatial structure, a feature that rather relied on the frequency information of a gesture was developed. Here follows a description of the processing procedure of the sparse data. If, again,

$$y_{\text{sparse}}(t_f, t_s, d), \quad t_f = 0, \dots, N_{t_f} - 1, \quad t_s = 0, \dots, N_{t_s} - 1, \quad d = 6m, \quad m \in \mathbb{Z}^+, \quad (5)$$

denotes the sparse signal, where t_f is the frame index, containing a number of sweeps, indicated by t_s , and d the distance to the sensor. Here, the frame and sweep update rates were chosen so that the time between two frames should be small. With a sweep rate of 2500 Hz and a frame rate of 18 Hz, the distance between two frames is $\frac{1}{18} - \frac{128-1}{2500} = 4.76$ ms. One frame covers $\frac{128-1}{2500} = 50.8$ ms.

In Figure 9, an example of the raw sparse signal, $y_{\text{sparse}}(t_f, t_s, d)$, is plotted.

Now, as one can notice on the phase shifts in the sparse signal, the frequency content is changing throughout the signal and this might be informative for the gesture. A standard transformation of time dependent data to capture frequency characteristics is the Discrete Fourier transform, DFT, defined as

$$Y(\ell) = \sum_{t=0}^{N-1} y(t) e^{-\frac{i2\pi}{N} \ell t}, \quad \ell = 0, \dots, N - 1.$$

As the frequency content of the signal (originated from the movement of the fingers) changes throughout the realization of the gesture, it is not interesting to transform the signal of the full gesture. Rather, one would want to see how the frequency content changes over time, and for this reason, a *Short Time Fourier Transform* (STFT) is applied to the data, separately for each frame

$$Y(t_f, \ell, d) = \sum_{t_s=0}^{N_{t_s}-1} y(t_f, t_s, d) w(t_s - \ell) e^{-\frac{i2\pi \ell t_s}{N_{t_s}}}, \quad \ell = 0, \dots, N_{t_s} - 1, \quad \forall t_f = 0, \dots, N_{t_f} - 1,$$

where $w(t)$ is a *Hanning window*, [15], to reduce leakage. The result being complex, the absolute value of the result is taken, yielding the square root of the *distance-dependent spectrogram*

$$S_y(t_f, \ell, d) = |Y(t_f, \ell, d)|^2.$$

One could choose to stop here and use the spectrogram as it is at this point. However, an additional last processing step was considered with two reasons in mind. First, the dimensionality is quite high, specifically for the IQ data. Second, the feature contains information of where in space the movement has occurred, which is not necessarily a good thing. Rather, it could be advantageous to be noticing no difference in the feature data when performing a gesture at, say, 5 or 15 cm. Therefore, the final form of the feature was set to the *distance-independent spectrogram*

$$S_y(t_f, \ell) = \sum_d \sqrt{S_y(t_f, \ell, d)} = \sum_d |Y(t_f, \ell, d)|.$$

Figure 11 illustrates an example of the resulting spectra. Note that the sparse signal is real valued, so its Fourier transform will be symmetric around 0. Hence, only the positive frequencies are kept and shown in the picture. It is further worth noting that this also implies it being impossible to distinguish between positive and negative velocities, that is, movements towards or away from the sensor.

Figures 9, 10 and 11 illustrates all steps of the whole processing chain:

$$y_{\text{sparse}}(t_f, t_s, d) \longrightarrow S_y(t_f, \ell, d) \longrightarrow S_y(t_f, \ell).$$

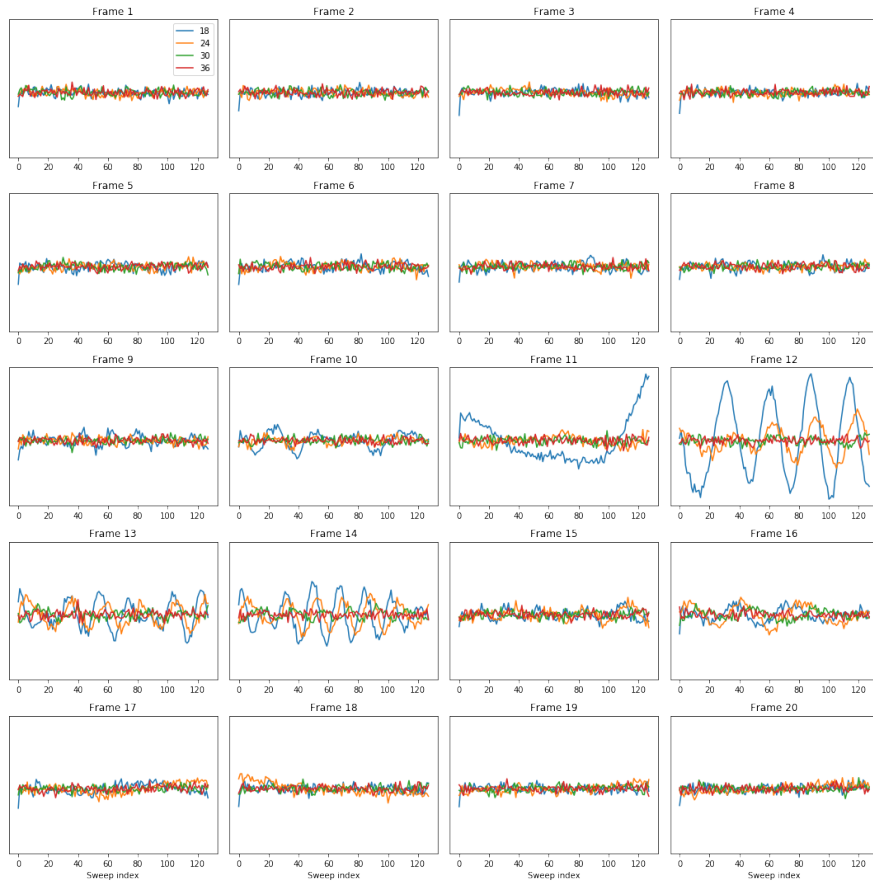


Figure 9: Example of the signal of a gesture (the swipe)

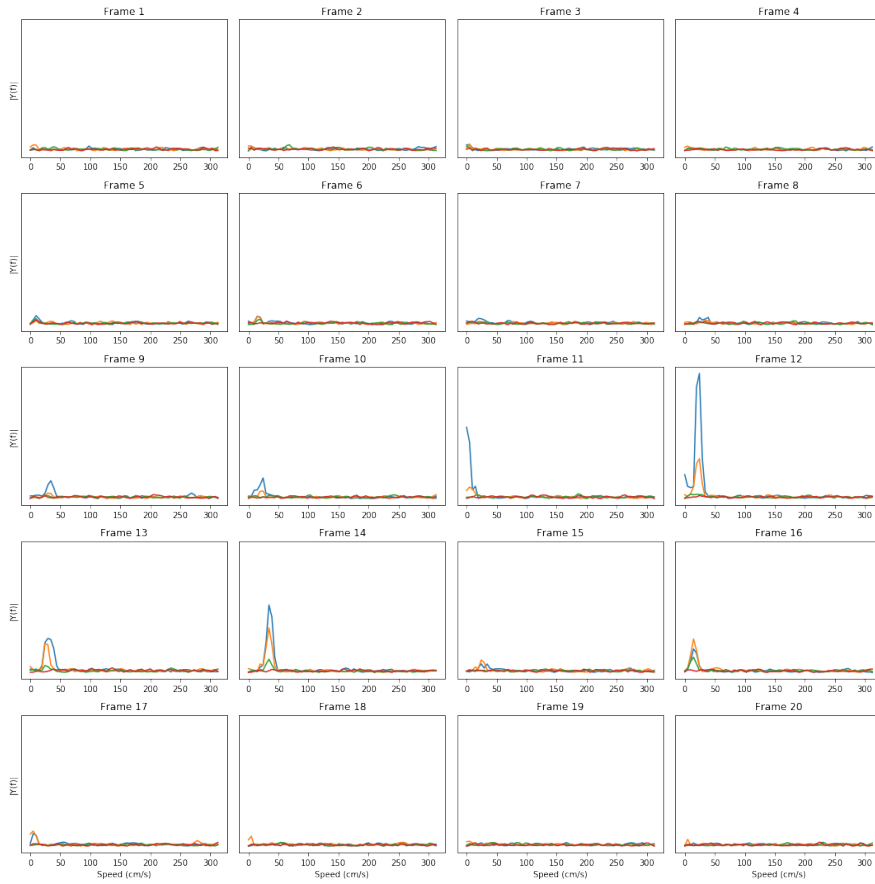


Figure 10: Distance dependent spectrogram of the sparse signal in Figure 9.

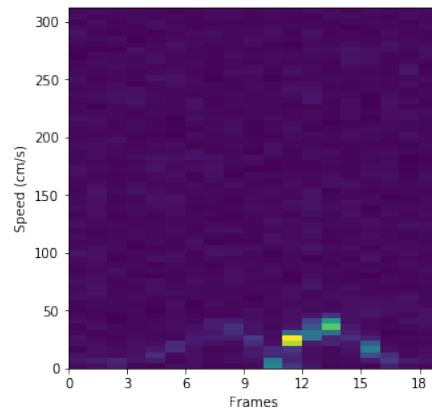


Figure 11: Distance independent spectrogram of the signal in Figure 9

Examples of final features of each class can be seen in Figure 13 on page 27.

When generating features from the IQ demodulated data in Dataset 2, one would want to apply a similar processing to obtain time-frequency informative features. Some details need to be taken into account in this case. Recall the notation of the IQ signal,

$$y_{\text{IQ}}(t_s, d), \quad t_s = 0, \dots, N_{t_s} - 1, \quad d \in \{d_0, \dots, d_{N_d-1}\},$$

where $d_0 = 13$ cm, $d_{N_d-1} = 35$ cm, and $d_{i+1} - d_i \approx \frac{6}{62}$ cm; $N_{t_s} = 186$. It is natural to split the IQ sweeps in time frames corresponding to those of the sparse signal, although there is no reason for it to be the best frame length. With a sweep rate of 175 Hz of the IQ signal, a time frame of 50.8 ms is covered by $\frac{0.0508}{1/175} = 8.89 \approx 9$ sweeps. The IQ signal can therefore instead be expressed as

$$y_{\text{IQ}}(t_f, t_s, d) = y_{\text{IQ}}(N_{t_f} t_f + t_s, d), \quad t_f = 0, \dots, N_{t_f} - 1, \quad t_s = 0, \dots, \tilde{N}_{t_s} - 1 = \frac{N_{t_s}}{N_{t_f}} - 1,$$

$$d \in \{d_0, \dots, d_{N_d-1}\}.$$

Thereafter, the same processing steps as described above were applied. One difference in the result with the IQ signal is that it is mixed with $e^{-i2\pi f_d t}$. Thus, it is complex and will have an asymmetrical Fourier transform. Therefore, it will be possible to distinguish the movements that have been performed towards and away from the sensor, which is not possible when using the sparse data. In subsection 6, Figure 14, given on page 28 contains examples for the IQ spectrogram feature of the considered classes.

4.1.1 Hyperparameters

There are possibilities to do other choices during the procedure of collecting the data and processing it that can be made. For example, investigations were made on the length N_{t_f} of a frame t_f . When creating the spectrogram feature, there is a trade-off between the resolution in time and the resolution in frequency. A longer frame implies a Fourier transform of more frequency bins, i.e., a higher resolution in frequency. Contrarily, the time resolution is diminished, since the full gesture is then divided into fewer parts. Therefore, different lengths were used to create features and they were compared, see Table 3.

Table 3: Different values of sweeps per frame, N_{t_f} , that were compared.

Sparse	IQ
64	7
128	9
256	11
	13
	15
	17

Another thing that was considered was a reduction of dimensionality of the sparse feature. As can be seen in Figure 11, it spans frequencies that the gesture does not contain, which was the case for all classes. Therefore, we attempted an idea of cutting the upper part of the spectrogram, to instead of a maximum speed of 312.5 cm/s, using only 141.5 cm/s, roughly halving the dimension. It was compared to not cutting the feature at all.

For all hyperparameter comparisons, one evaluation was done using a KNN and one using a 2D CNN classification. These methods are introduced in subsection [5](#).

4.2 Impulse Response Feature

Another, slightly more involved, modelling approach was also examined. Let the recorded radar data when doing a gesture, the reflection of the emitted signal, x , be modeled as a filtration of x ,

$$x(t) * h(t).$$

The idea is then that the (sequence of) impulse responses of the filter could serve as a characteristic feature of the gesture. It thus relies on the ability to estimate impulse responses, knowing the emitted signal $x(t)$ and the recorded signal. Due to the fact that the radar sensor matches the propagated signal with an internal signal, the recorded raw signal may be modeled as $x(t) * x(t)$. On top of that, several processing filters are used, yielding a recorded signal that may be well modeled as

$$y(t) = x(t) * h(t) * x(t) * c(t) + v(t) = z(t) * h(t) + v(t), \quad (6)$$

where $v(t)$ is an additive noise, here modelled as a white Gaussian process. Thus, knowing $y(t)$ and $z(t)$, one can estimate $h(t)$. This problem could equivalently be expressed as

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} z_0 & 0 & \dots & 0 \\ z_1 & z_0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ z_N & z_{N-1} & \dots & z_0 \\ z_{N+1} & z_N & \dots & z_1 \\ \vdots & \vdots & \ddots & \vdots \\ z_M & z_{M-1} & \dots & z_{M-N} \\ 0 & z_M & \dots & z_{M-1-N} \\ 0 & 0 & \dots & z_{M-2-N} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & z_M \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_N \end{bmatrix} + \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_N \end{bmatrix}, \quad \text{or } \mathbf{y} = \mathbf{Z}\mathbf{h} + \mathbf{v}.$$

A first approach would be to estimate \mathbf{h} by least squares (LS), but additional regularization approaches have been considered.

4.2.1 Elastic net

One regularized version of the least squares estimation can be formed including simultaneously L^1 and L^2 penalties, such that

$$\min_{\mathbf{h}} (||\mathbf{y} - \mathbf{Z}\mathbf{h}||_2^2 + \lambda(\alpha||\mathbf{h}||_1 + (1 - \alpha)||\mathbf{h}||_2)), \quad \lambda \in \mathbb{R}, \alpha \in [0, 1].$$

The resulting estimator yields a sparser solution than the LS estimator, being regularized to cope with bad conditioned problems. The resulting estimator is the so-called *Elastic net*. In this work, a pre-implemented algorithm in the *Scikit-learn* package was used. From each IQ sweep, one impulse response was estimated. The elastic wideband feature therefore consisted of a 2D-matrix: one dimension ordering the sweeps during the gesture duration, one running

along the time delay axis of the filter impulse response. An example of a feature can be seen in Figure 15a.

4.2.2 A sparser solution by wideband integrated dictionaries

In [17], an algorithm for estimating sparse impulse responses was successfully employed. The problem examined in this work was even more constrained, with a goal of estimating both the impulse response and the transmitted waveform, only knowing the received signal. The idea that was adopted in this work is that of using a wideband integrated dictionary. Then, concretely, in the frequency domain, the frequency band $(0,1]$ of a channel of initially Q points, f_1, \dots, f_Q , is replaced by a dictionary of integrated elements

$$w_q(t) = \int_{f_q}^{f_{q+1}} z(t, f) df. \quad (7)$$

One reason for this change, in the original setting, is to be able to capture all the frequencies which are not on-grid and therefore obtain a more precise channel estimate. The resulting iterative process consists of successively estimating activated widebands, followed by refining the grid in those widebands, resulting in a "zooming" of the employed dictionary. In this work, the wideband approach was not used iteratively but only once, with the aim of achieving some level of sparsity. The algorithm implemented in this work was as follows:

Algorithm 1: Sparse estimation of filter impulse response using Wideband Integrated Dictionary and Elastic net.

Result: Returns a compact estimation $\hat{\mathbf{h}}_{\text{compact}}$ of \mathbf{h} in (6), consisting of the estimation over the k most active widebands, together with information of which the widebands are.

for $\mathbf{y}(t), t = 1, \dots, N_{t_s}$ **do**

Form wideband dictionary \mathbf{W} as in (7);

Normalize \mathbf{y} and \mathbf{W} ;

Separate $\mathbf{y}_{\text{sep}} = [\text{Re } \mathbf{y}, \text{Im } \mathbf{y}]$ and $\mathbf{W}_{\text{sep}} = [\text{Re } \mathbf{W}, -\text{Im } \mathbf{W}; \text{Im } \mathbf{W}, \text{Re } \mathbf{W}]$;

Find the k most active widebands by solving $\hat{\mathbf{g}} = \arg \min_{\mathbf{g}} \|\mathbf{y}_{\text{sep}} - \mathbf{W}_{\text{sep}} \mathbf{g}\|_2$ and

picking $\hat{\mathbf{I}}_{\text{active}} = \text{argsort}(\hat{\mathbf{g}})[1 : k]$;

Form a dictionary of active widebands, $\mathbf{W}_{\text{active}} = [\mathbf{W}_i]$, for $i \in \hat{\mathbf{I}}_{\text{active}}$;

Separate $\mathbf{W}_{\text{active}}$;

Solve $\hat{\mathbf{h}}_{\text{wb}} = \arg \min_{\mathbf{h}} \|\mathbf{y}_{\text{sep}} - \mathbf{W}_{\text{sep}} \mathbf{h}\|_2^2 + \lambda(\alpha \|\mathbf{h}\|_1 + (1 - \alpha) \|\mathbf{h}\|_2)$;

Add information of which the active widebands are, $\hat{\mathbf{h}}_{\text{compact}} = [\hat{\mathbf{I}}_{\text{active}}, \hat{\mathbf{h}}_{\text{wb}}]$

end

An example of an estimation can be seen in Figure 16a.

5 Classification Algorithms and Model Selection

5.1 k-Nearest Neighbour

As a benchmark classification performance, the simple KNN classifier is used, see [18] for a deeper introduction to this classifier. It does not formally impose a model on the data; it simply finds the k training points $x_i \in N_k(x)$ closest in distance to the point x that is to be classified, and chooses its class $\hat{y}(x)$ as the majority vote of the training points' class belongings $y(x_i)$, as

$$\hat{y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y(x_i).$$

However, there are still parameters to set: the number of neighbours k and the distance measure to be used. Typically, a higher k implies a smoother decision boundary. In this work, the classification accuracy was tuned for $k = 1, \dots, 14$. For the metric, the L^p -norm

$$d(x, y) = \left(\sum_i |x_i - y_i|^p \right)^{1/p},$$

was considered, for $p = 0, 1, 2$. For $p = 2$, this is the Euclidian distance, and for $p = 1$ it is distance in the L^1 norm.

5.2 Convolutional Neural Networks

In this section the neural network model will be briefly described, most notably to define the terminology and to discuss the choice of architecture. For a thorough explanation of neural networks, see, for instance, [19].

A regular, fully-connected, neural network is a model which connects an input vector to, another one-dimensional, output vector. It does this through a chain of the so-called *layers*, where the input vector to that layer, $\mathbf{x}^{(k)}$, undergoes a linear transformation, $\mathbf{z}^{(k)} = \mathbf{W}^{(k)} \mathbf{x}^{(k)} + \mathbf{b}^{(k)}$, before being fed to a non-linear function, f , yielding the output vector of that layer, $\mathbf{x}^{(k+1)} = f(\mathbf{z}^{(k)})$. Figure 12a illustrates a simple network architecture. The units in each layers are referred to as *nodes*, and the connection between the nodes can be seen as an illustration of the linear transformations between two layers. They are called *weights* and are gathered in the weight matrix $\mathbf{W}^{(k)}$. The vector $\mathbf{b}^{(k)}$ is called *bias* and constitute, together with the weight matrix, the parameters of the layer. The non-linear function f is also called *activation function*, which, in this work, is either the *Rectified Linear Unit transformation* (ReLU)

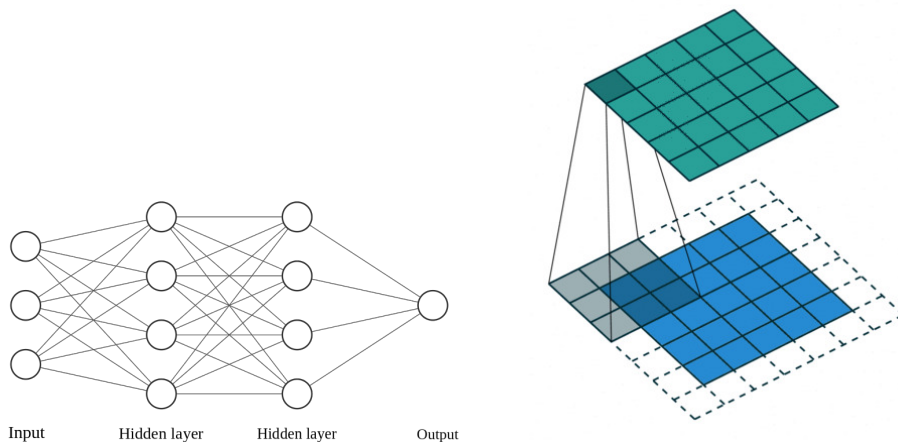
$$f(x) = \max(0, x)$$

or the *softmax transformation*

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}, \quad \forall x_i \in \mathbf{x}^{(k)}.$$

The softmax activation has the property that it scales the outputs to values in $[0, 1]$, which can thus be interpreted as a probability.

The feature design described in this work results in a 2D data matrix, formally of one frequency dimension and one time dimension. The use of 2D images is a very classic feature format for which there exist well developed neural networks models. The standard network for an image classification task is the Convolutional Neural Network (CNN), which learns to recognize patterns in the image by sliding a kernel over the picture. As [19] puts it, CNNs are specialized to "work with data that has a clear grid-structured topology and to scale such models to very large size". One of the important advantages of this kind of model is namely that it contains fewer parameters than fully connected networks, as images easily can be of very high dimension.



(a) Illustration of a fully connected neural network with two hidden layers.

(b) Illustration of a convolutional layer with a single filter. A kernel, here 3×3 , slides over the zero-padded image (in blue) and is convolved with the overlap of the input picture at each position. The result is a new 2D "image" (in green). For a higher number of filters, the same number of kernels is introduced, each having different parameter values. Each kernel gives rise to an output image. The picture is taken from [20].

Figure 12

The standard 2D CNN convolves local parts of the image with a rectangular, or even square, kernel over the picture. In principle, it therefore detects patterns at any location in the picture, no matter the x and y coordinate of such a pattern. Figure 12b illustrates this procedure. Recall that the feature image in this work is composed of one frequency dimension and one time dimension. A certain gesture is expected to exhibit a certain pattern in frequency, but it could occur at any time within the time frame of the recording. An approach to make a model keep track of these conditions could thus be by using a 1D CNN. This network also slides a kernel over the picture, but only in one direction (the time dimension). The idea is that it would look for a specific pattern in frequency, but anywhere in time.

Two types of dimensionality reductions and regularizations are applied in this work. One is the *max-pooling*, which downsamples the feature image by only keeping the maximum element among several in a defined region of (2×2) pixels. This operation thus reduces the dimensionality and therefore permits a reduction of parameters in the model. Another regularization is the

so-called *dropout*, which is defined by a probability p . All nodes at the dropout layer is dropped with probability p and correspondingly kept with probability $1 - p$.

In all neural networks modelling tasks, it is an issue of selecting an appropriate architecture and of choosing the values of hyperparameters. In this work, all architectures of type

$$\text{Input} \longrightarrow [(\text{Conv} + \text{ReLU}) \cdot N \longrightarrow \text{Maxpool}] \cdot M \longrightarrow [\text{FC} + \text{ReLU} + \text{Dropout}] \cdot K \longrightarrow \text{FC} + \text{Softmax}, \quad (8)$$

were employed, where N, M , and K are integer parameters, defining how many times the corresponding patterns should be repeated. Their ranges are defined in Table 4. Conv is the convolution layer, where the input images are zero-padded to keep the dimension throughout the convolutional layers. The sequence of convolutional and max-pooling layers processes the original image to a set of smaller feature "images". These are reshaped into a one-dimensional vector, permitting fully connected (FC) layers, as in Figure 12a. The first contains 100 nodes, the second 5, for the number of classes. To give an example, if $N = 1, M = 1, K = 0$, then the network consists of one convolutional layer, one maxpooling and then the output, fully-connected, layer. By this approach, the architecture design is limited to a set of scalar hyper parameters. A random search approach was employed, where possible choices within the architecture was defined, a descent amount of combinations, 100-1000, depending on the task, were drawn at random and evaluated. For computational reasons, a limit of parameters in the model above 10^5 was discarded. Each architecture was trained for 20 epochs (updates of network parameters) on the training set and evaluated on the validation set. The two models that performed the best were then retrained, this time using cross validation (see Section 5.4), on the concatenation of the training and validation set. In case of a test evaluation, the best architecture was retrained on the full training set, before being evaluated on the test set.

Table 4: Ranges for the parameters in the CNN architectures. Random selection of all parameters were realized and evaluated in order to find an appropriate model.

	N	M	K	Kernel size	Number of filters	Dropout rate	Optimizer	Batch size
2D CNN	1-4	1-3	0-1	(3,3), (4,4)	16, 32, 64	[0, 0.5]	Adam, RMSprop, SGD	32, 64, 128
1D CNN	1-4	1-3	0-1	(30,1)	16, 32, 64	[0, 0.5]	Adam, RMSprop, SGD	32, 64, 128

Note that after a 1D convolutional layer, the volume of input shape (n, m) will become (l, m) , where $l =$ the number of filters. Therefore, on subsequent convolutional layers, a kernel size of $(l, 1)$, was used.

5.3 Complexity Analysis

In this section follows an investigation of the computational complexity of preparing the feature from data as well as predicting its class belonging, for all different classification algorithms.

These formulas were evaluated for some models and datasets, and are presented in Table 15 on page 37.

5.3.1 Feature Preprocessing

Here, the preprocessing of the spectrogram feature is considered, apart from the processing performed in the hardware, e.g. the IQ demodulation. During the process, the data needs to be transformed by a Fast Fourier Transform (FFT). The longtime known algorithm to do this, introduced by [21], requires less than $2N \log_2 N$ operations, when the vector is of length N . The arithmetic complexity of the preprocessing is derived using this estimate. Let the raw data of a full gesture have shape (N_{t_f}, N_{t_s}, N_d) . Then, the FFT is applied several times, along the second axis. This step therefore requires $N_{t_f} N_d 2N_{t_s} \log_2 N_{t_s}$ operations. For the sparse feature, having symmetric DFT, the dimension is at this point $(N_{t_f}, \frac{N_{t_s}}{2}, N_d)$; the IQ feature has intact shape. These are finally transformed by integrating over the last axis, i.e., the distance dimension. This requires $N_{t_f} \frac{N_{t_s}}{2} (N_d - 1)$ additions for the sparse feature and $N_{t_f} N_{t_s} (N_d - 1)$ for the IQ feature. The total amount of operations in the preprocessing of the data is thus

$$\begin{aligned} N_{t_f} N_d 2N_{t_s} \log_2 N_{t_s} + \frac{N_{t_f} N_{t_s} (N_d - 1)}{2} \quad (\text{sparse}), \\ N_{t_f} N_d 2N_{t_s} \log_2 N_{t_s} + N_{t_f} N_{t_s} (N_d - 1) \quad (\text{IQ}). \end{aligned} \tag{9}$$

Asymptotically, this number amounts to $\mathcal{O}(N_{t_f} N_{t_s} N_d \log N_{t_s})$.

5.3.2 KNN

Let the feature have dimensionality (m, n) . In the worst case of a k-Nearest Neighbour search, in the case of brute force search, the distance between two points of dimension d can be computed in $\mathcal{O}(d)$ time. The full sorting of a set of cardinality p , requires therefore $\mathcal{O}(dp)$ [22], or $\mathcal{O}(mnp)$.

5.3.3 CNN

Again, let the input image have dimensions (m, n) . Let the kernel size be (k, k) , for the 2D CNN. In the case of 1D CNN, the initial kernel has size $(n, 1)$, followed by $(l, 1)$, l being the number of filters. A detailed derivation of the complexity is presented in Appendix B, on page 45. For the 2D CNN architecture considered,

$$\begin{aligned} \mathcal{O}(k^2 N m n l (1 - 4^{2-M})) + \mathcal{O}(m n l (1 - 4^{2-M})) + \mathcal{O}\left(\frac{m n l}{4^M} (1 + K)\right) + \mathcal{O}(K) \quad \text{operations}, \\ \mathcal{O}(N m n l (1 - 4^{3-M})) + \mathcal{O}(K) \quad \text{calls to activation functions}, \end{aligned} \tag{10}$$

are required, according to equation (13) on page 46. For the 1D CNN models, this number is

$$\begin{aligned} \mathcal{O}(m l (n - 1)) + \mathcal{O}(m l N (1 - 2^{1-M})) + \mathcal{O}(m l (1 - 2^{1-M})) + \mathcal{O}\left(\frac{m l}{2^M} (1 + K)\right) + \mathcal{O}(K) \quad \text{operations}, \\ \mathcal{O}(N m l (1 - 2^{2-M})) + \mathcal{O}(K) \quad \text{calls to activation functions}, \end{aligned} \tag{11}$$

according to equation (14) on page 47.

5.4 Cross-Validation

As explained above, choices of parameters and even architectures are done to make the model as explanatory of the data as possible. The standard approach for choosing the optimal values of the parameters is to divide the data available into three parts: a training set to which the model is fit; a validation set on which the fitted model is evaluated, for different choices of model parameters; a test set for which the final model is evaluated, to compare different models between each other. In this work, the sets were distributed as: training set 70%, validation set 15%, and test set 15%. This split was employed when searching for an architecture: different configurations were fit using the training data and then compared using the validation data.

To take as much advantage of the data as possible, cross-validation can be employed [18]. Instead of doing a single split between train and validation data, their union is split into k , obtaining k -fold cross-validation. Each of the k parts acts as the validation set once, when the others act training set, to which the model is repeatedly fit. The validation evaluation is then assessed through an average of all validation results, and the sample standard deviation quantifies the uncertainty of the prediction score. In this work, $k = 5$.

For Dataset 3, originated from several people, the Leave-One-Out version of the cross-validation was likewise used. To investigate how well a fitted model generalizes to data from an unseen person – which would be the case of interest when developing a product – the test folds were chosen to consist of all the data recorded by 2-4 people and the train data, to which a model was fit, of the remaining 14-16 persons. The number of people in the test set was allowed to vary, since people had gathered different amounts of data. Note though that since each person had gathered the same amount of data from each class, the distribution of the classes in the test sets was the same. The people constituting the test set was matched such that their data should add up to approximately the same size, being 15% of the full dataset. The remaining 85% constituted the training set, which was itself split into 5 folds, for model selection. The test result from this cross-validation can accordingly provide information of how well the model generalizes to an unseen person.

Table 5: The distribution of the data over the 18 people, and specification of the test sets in the special case of cross-validation.

Person	Size	Testset	Total size
1	808	1	1108
2	300		
3	806	2	1106
4	300		
5	302	3	1202
6	600		
7	300		
8	502	4	1002
9	200		
10	200		
11	100		
12	200	5	1204
13	202		
14	600		
15	202		
16	510	6	1110
17	400		
18	200		

6 Results

6.1 Feature Extraction

Figures 13, 14 show examples of the spectrogram features from Dataset 1 and 2, whereas Figures 15a and 16a show examples of the impulse response features for the elastic net and the wideband estimation.

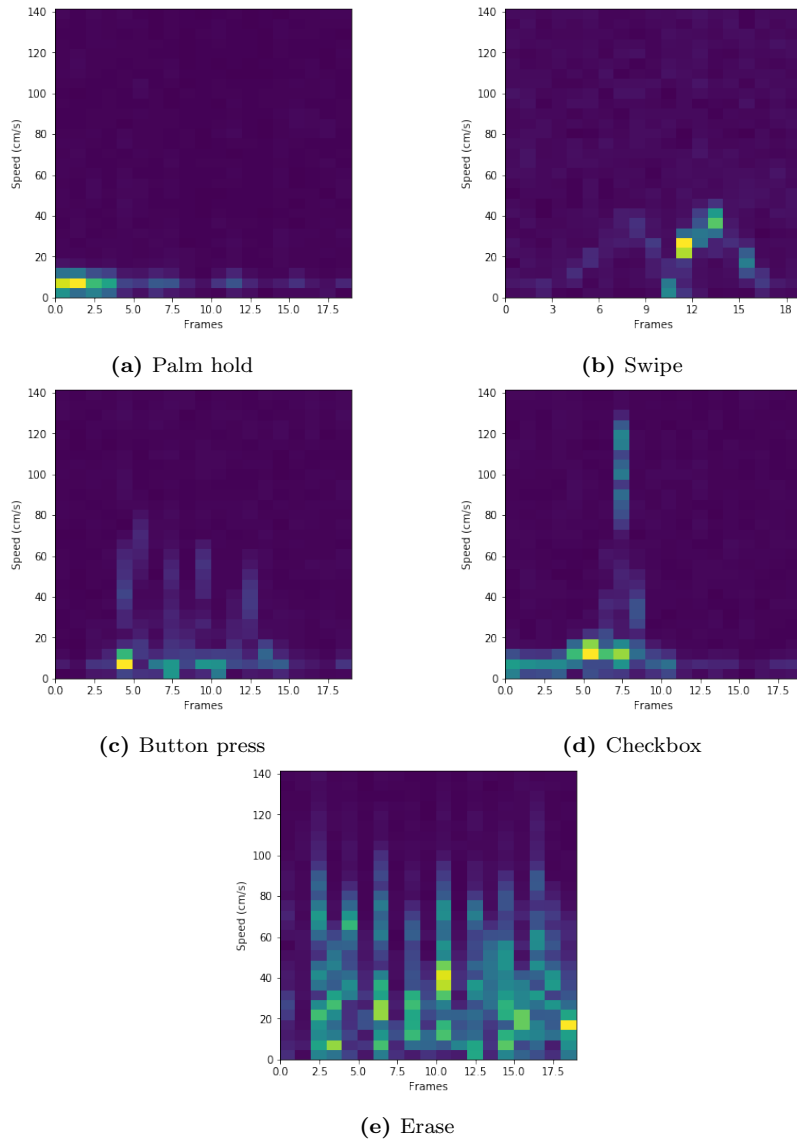


Figure 13: An example of each class, using the sparse distance independent spectrogram feature.

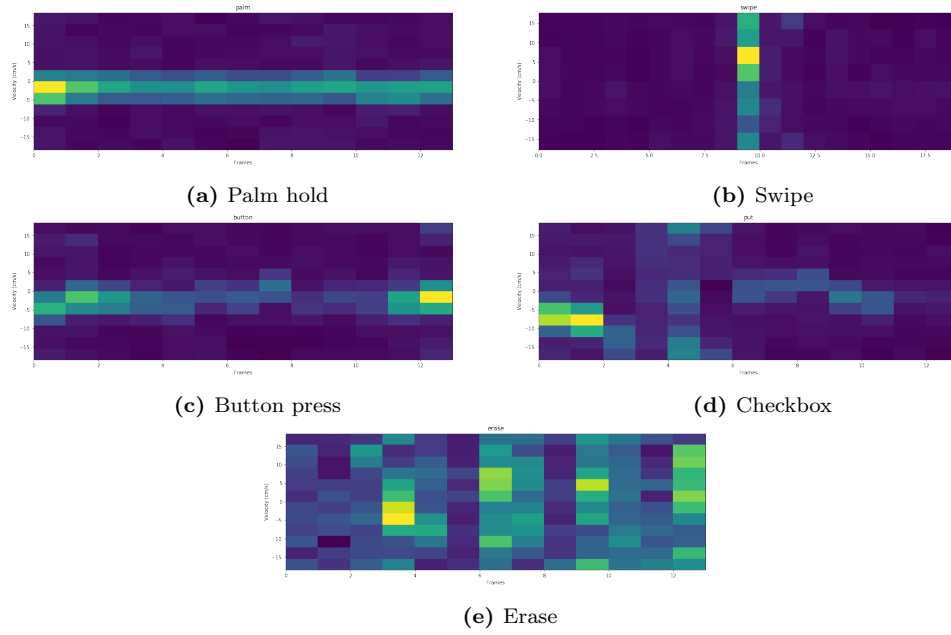
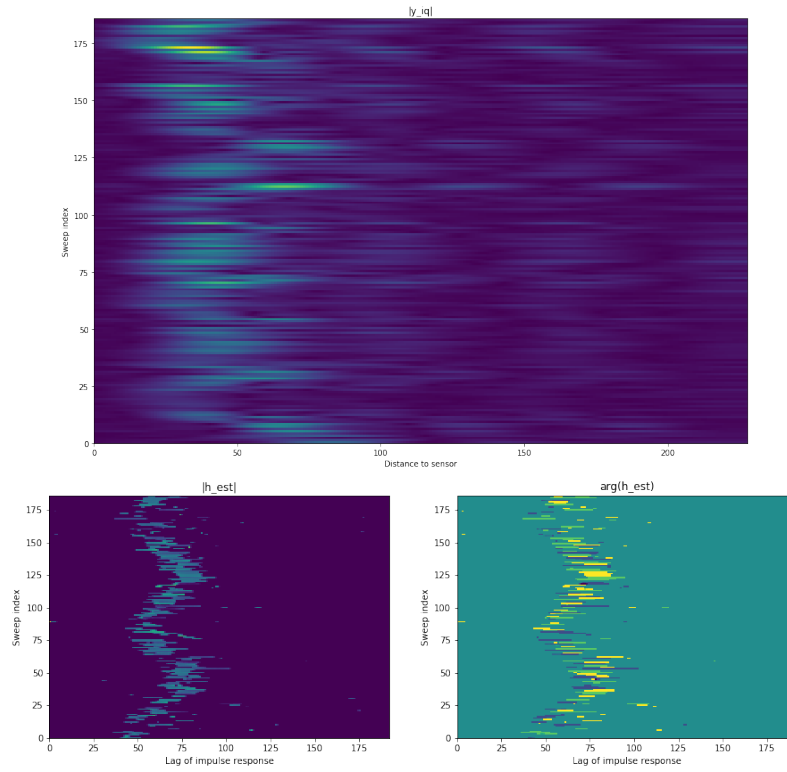
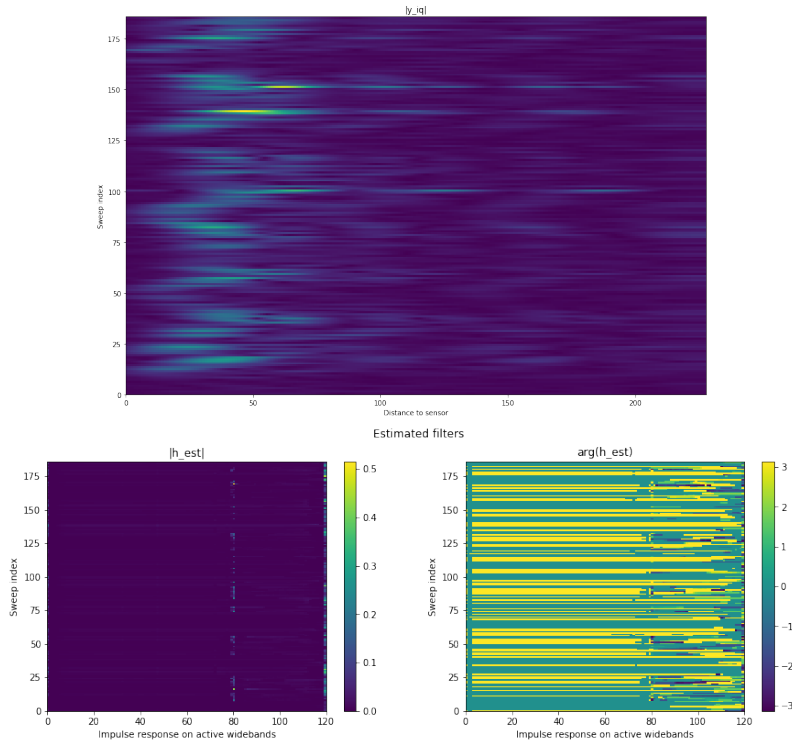


Figure 14: An example of each class, using the IQ distance independent spectrogram feature.



(a) Elastic net estimation of an example of Erase.

Figure 15: Elastic net estimations.



(a) Wideband integrated dictionary estimation of an example of Erase.

Figure 16: Elastic net estimations.

The validation results for feature extraction investigations can be found in Table 6-8, both using KNN and 2D CNN as classifiers. All validation results are obtained using a 5-fold cross-validation, as discussed in Subsection 5.4. In Table 6, the results using the IQ data is presented, using features of frames of different lengths. According to it, KNN prefers 17 sweeps/frame, whereas 2D CNN performs best for 13 sweeps/frame.

Table 6: IQ feature extraction. For 2D CNN, 100 randomized architectures were compared. The two best were retrained during 20 epochs and compared using 5-fold cross-validation.

Sweeps/frame	KNN	2D CNN
7	0.90382 (\pm 0.01038)	0.97989 (\pm 0.00524)
9	0.91641 (\pm 0.00571)	0.98478 (\pm 0.00218)
11	0.92865 (\pm 0.00767)	0.98653 (\pm 0.00118)
13	0.93547 (\pm 0.00560)	0.98689 (\pm 0.00379)
15	0.94404 (\pm 0.00454)	0.98566 (\pm 0.00532)
17	0.95016 (\pm 0.00663)	0.98444 (\pm 0.00324)

In Table 7, the results for the sparse data can be seen. When it comes to sweeps per frame, KNN performs preferably when using 64 sweeps, whereas 2D CNN should have features formed using 128 sweeps. Furthermore, the accuracy is slightly lower when not cutting off the high frequencies from the feature.

Table 7: Sparse feature extraction. For 2D CNN, 100 randomized architectures were compared. The best was retrained during 20 epochs using 5-fold cross-validation.

Sweeps/frame	KNN	2D CNN
64	0.93709 (\pm 0.00447)	0.99498 (\pm 0.00166)
128	0.93350 (\pm 0.00372)	0.99534 (\pm 0.00154)
256	0.87023 (\pm 0.00419)	0.97114 (\pm 0.00527)
64 (no freq. cut)		0.99355 (\pm 0.00280)
128 (no freq. cut)		0.99498 (\pm 0.00238)
256 (no freq. cut)		0.97096 (\pm 0.00588)

Finally, the features based on impulse response estimates should be treated. In Table 8, validation results when using the impulse response estimates can be found, the performance obtained using KNN classification. The two estimators are compared, and it is investigated whether to keep the full information of the complex estimates, $(|h|, \arg(h))$, or if one should use $|h|$. For the Elastic net estimates, adding information of the complex part leads to a drastic decrease in accuracy. For the Wideband estimates, there is no difference, and the accuracy is approximately the probability of guessing correctly. In Table, 9, the dimensions on the extracted features are given.

Table 8: Feature extract evaluations, classified by KNN, using impulse response estimates as features.

	$\text{abs}(\mathbf{h})$	$(\text{abs}(\mathbf{h}), \text{arg}(\mathbf{h}))$
Elastic net	0.77684 (\pm 0.00898)	0.33280 (\pm 0.03875)
Wideband	0.20269 (\pm 0.02078)	0.20269 (\pm 0.02078)

Table 9: Dimensions of features chosen.

Feature	Dimension
IQ spectrogram	(14,13)
Sparse spectrogram	(20,30)
Impulse response (elastic net)	(186, 191)
Impulse response (wideband)	(186, 123)

6.2 Classification Results on Dataset 1 and 2

In Table 10, the test results of all features, using different classifiers, are given. The KNN algorithm had its highest accuracy for the IQ spectrogram feature; the 1D CNN model performed best on the sparse version; so did the 2D CNN. These models will be referred to as *Model A* (KNN), *Model B* (1D CNN), and *Model C* (2D CNN) further on. How well the best performing model, *Model C*, predicted each of the classes can be assessed through a confusion matrix, which summarizes the predictions and the true classes in a table. This provides separate accuracy estimates for all classes and further information between which classes there are confusions. Figure 17 contains the confusion matrix for *Model C*.

Table 10: Test accuracies for different features using KNN, 2D CNN, and 1D CNN models. For the convolutional neural networks, 1000 architectures were drawn and trained for 20 epochs and evaluated on validation data. The two best were retrained using 5-fold cross-validation, and the best among them was retrained on all train data and evaluated on test data.

Feature	KNN	2D CNN	1D CNN
IQ	0.95545	0.98416	0.96040
Sparse	0.93198	0.99594	0.99391
Impulse response (wideband)	0.20297		
Impulse response (elastic net)	0.76337		

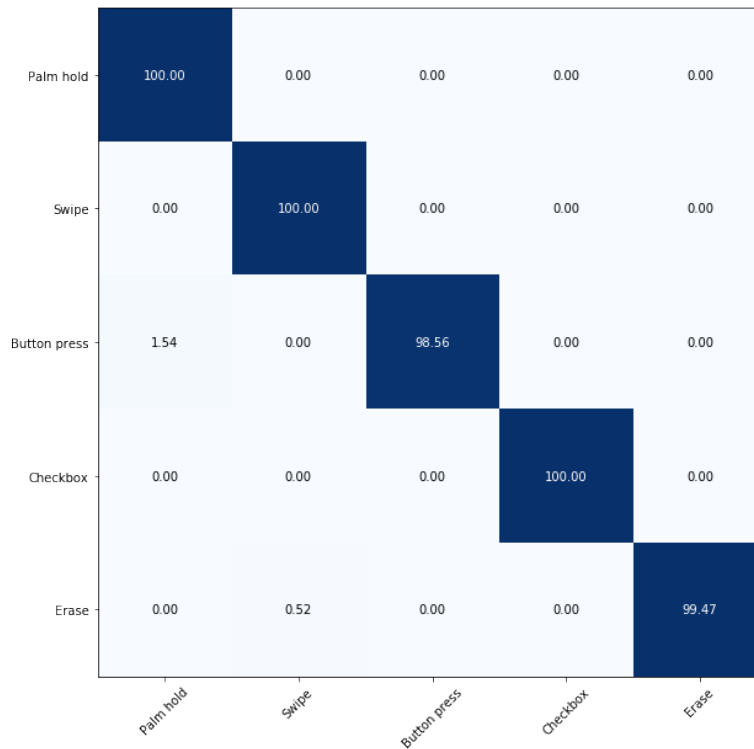


Figure 17: Confusion matrix for *Model C*, on the test set of Dataset 1. The x-axis indicates the prediction and the y-axis the true class belonging.

6.3 Classification Results on Dataset 3

Next, the results using Dataset 3, that was recorded by 18 different people, are presented. No feature extract was performed on this dataset, and consequently the feature is the same as was chosen on Dataset 1.

6.3.1 Test Results 5-fold Cross-Validation

Table 11 summarizes the result when testing the model on a randomly chosen subset of all 18 people, on which the model was trained, applying a 5-fold cross-validation. The test data was therefore recorded by people that also recorded the training data, i.e., the models have already seen these people. The model with the highest accuracy was again 2D CNN.

Table 11: Model evaluations for Dataset 3, originated from 18 people. The features, and its hyperparameters, were the ones chosen using Dataset 1. That is, for KNN, framelength of 64 sparse sweeps, keeping the 15 smallest frequency bins; for CNN the framelength is 128, and 30 frequency bins are kept. 1000 drawn architectures were trained for 20 epochs and compared on validation data. The two best were retrained using 5-fold cross-validation over all people’s recordings. The best was retrained on all train data and evaluated on the test data.

	Test accuracy
KNN	0.86282
2D CNN	0.97614
1D CNN	0.95527

6.3.2 Test Results Leave-One-Out Cross-Validation

Here, the results using the approach of evaluating models on unseen people are presented. Recall that in this case, a [test set](#) i of 2-4 people was set aside. A model was trained on the remaining people’s data, i.e., all test sets except set i , and evaluated on test set i . Thus, this result gives information of how well an optimized model generalizes to unseen people. The test set was also fed to the models fit to Dataset 1. Recall that these models, denoted [Model A](#), [Model B](#), and [Model C](#) were trained on the data from only one person. Note that one person of test set 1 was the same person that recorded the data for Dataset 1, and therefore that Testset 1 is not unseen for any of these models. That test evaluation is therefore excluded from the mean, median, and standard deviation computations.

The result of the KNN classification can be found in [Table 12](#), for 1D CNN in [Table 13](#), and for 2D CNN in [Table 14](#).

Table 12: Evaluation of KNN models on only people whose data the model did not use during training.

Testset	Test accuracy	Test acc. on Model A
1	0.73375	(0.85560)
2	0.82098	0.67089
3	0.76955	0.53661
4	0.75050	0.54192
5	0.76661	0.58056
6	0.76126	0.50991
Mean	0.76711	0.56798
Median	0.76394	0.54192
Std	0.02687	0.05619

Table 13: Evaluation of 1D CNN models on only people who’s data the model did not see during training. For each test set, 100 architectures were drawn, trained for 20 epochs and compared. The two best were retrained using 5-fold cross-validation, and the best among them was retrained on all train data and evaluated on the test set. Note that [Model B](#) was trained on data that is in Testset 1, which is why that test accuracy was excluded from the computed mean, median, and standard error.

Testset	Test accuracy	Test acc. on Model B
1	0.91787	(0.93502)
2	0.95750	0.85624
3	0.89850	0.78785
4	0.91018	0.81238
5	0.94020	0.88704
6	0.92883	0.77477
Mean	0.92551	0.82366
Median	0.92335	0.81238
Std	0.01946	0.04214

Table 14: Evaluation of 2D CNN models on test data originated from people not being in the train set. For each test set, 100 architectures were drawn, trained for 20 epochs and compared. The two best were retrained using 5-fold cross-validation, and the best among them was retrained on all train data and evaluated on the test set. Note that [Model C](#) was trained on data that is in Testset 1, which is why that test accuracy was excluded from the computed mean, median, and standard error.

Testset	Test accuracy	Test acc. on Model C
1	0.94365	(0.96390)
2	0.97188	0.90958
3	0.93591	0.85191
4	0.94627	0.85329
5	0.96337	0.89452
6	0.94456	0.80000
Mean	0.95094	0.86186
Median	0.94541	0.85329
Std	0.01248	0.03832

Again, the 2D CNN classifier led to the best results. The performance of models [A](#), [B](#), and [C](#) on the unseen test data is clearly lower than for the models having seen data from more people. For the 2D CNN classifications, confusion matrices are also presented in Figures [18](#) and [19](#). The first one is an average of the six evaluations corresponding to the left column in Table [14](#), where one test set at a time is excluded from training and a model is fit to the remaining sets. The second one is an average of the evaluations on [Model C](#), corresponding to the right column of Table [14](#), using each of the test sets. Again, Test set 1 was excluded, as it contains data that was used to fit [Model C](#).

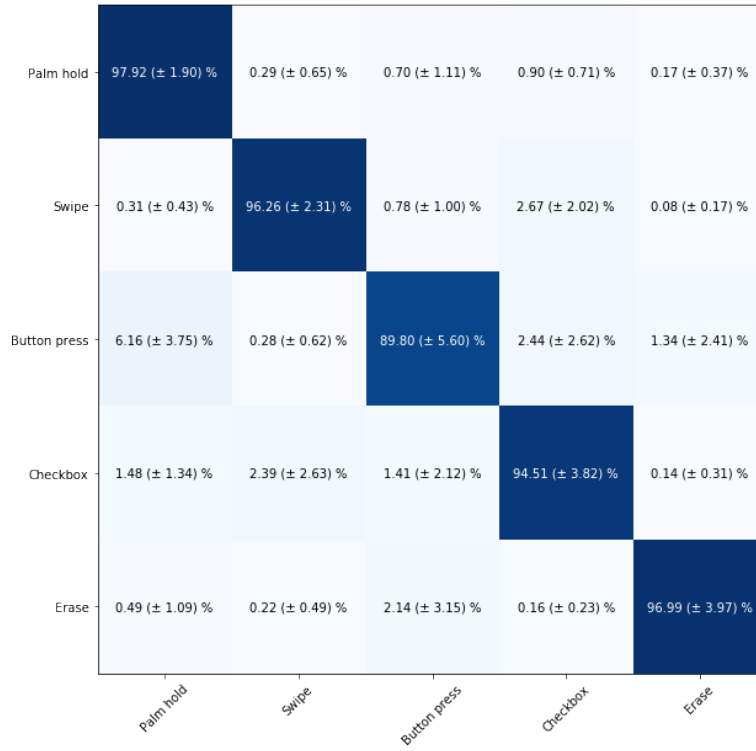


Figure 18: Confusion matrix using 2D CNN classifiers, for models trained on Dataset 3, by means of leave-one-out cross-validation. The x-axis indicates the prediction and the y-axis the true class. For each test set taken aside, the best architecture according to the training set was evaluated on the unseen test set and the corresponding confusion matrix calculated. These are the same models as in Table 14. The shown result is the average of the six confusion matrices, each originated from one test set.

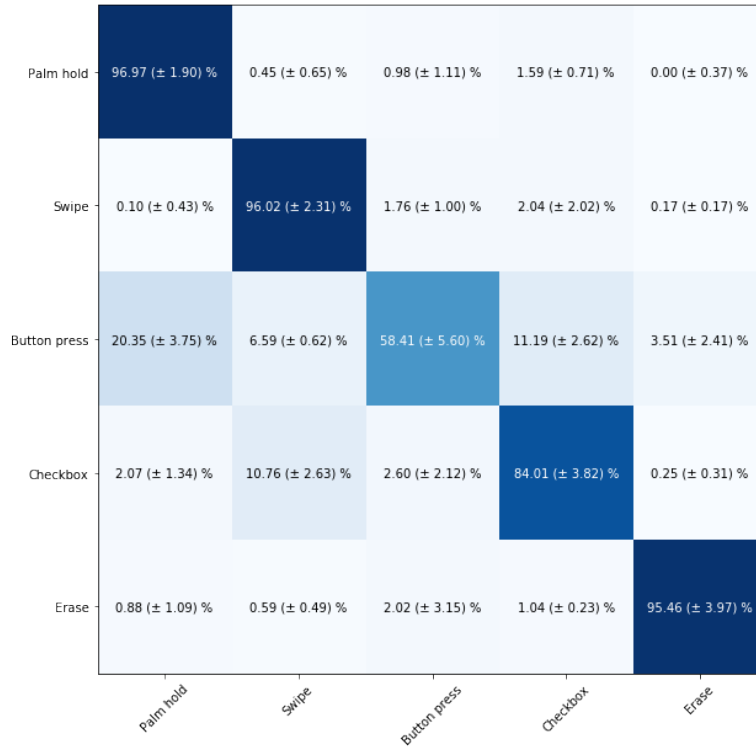


Figure 19: Confusion matrix using 2D CNN classifiers, where [Model C](#), which was trained on Dataset 1 (1 person), was tested on [Testset 2-6](#) from Dataset 3 (18 persons). A separate confusion matrix per test set was computed, and the presented result is the average confusion rates with standard errors. The x-axis indicates the prediction and the y-axis the true class.

6.4 Complexity Analysis

The computational complexity for preprocessing and prediction, introduced at page 23, was determined for the optimized architecture, to investigate another aspect of the classifiers. It is presented in Table 15, together with the sizes of the models, i.e., the number of parameters of the network. According to the table, prediction is considerably less computationally expensive for the 1D CNN models. However, an important computational cost is required for the preprocessing, increasing the total arithmetic complexity also for these models. The models performing best in accuracy, the 2D CNN for Dataset 1 and Dataset 3, are both the most computationally heavy and the largest models.

Table 15: Computational complexity of the final CNN models, counted in number of operations and evaluations of an activation function. The numbers are evaluated using (9) on page 24 for the preprocessing, (13) on page 46 for 2D CNN prediction, and finally (14) on page 47 for 1D CNN prediction. The size is the number of parameters of the model.

Model	Dataset 1		Dataset 2		Dataset 3	
	1D CNN	2D CNN	1D CNN	2D CNN	1D CNN	2D CNN
Operations Preprocessing	$2 \cdot 10^5$	$2 \cdot 10^5$	$4 \cdot 10^5$	$4 \cdot 10^5$	$2 \cdot 10^5$	$2 \cdot 10^5$
Operations prediction	$4 \cdot 10^3$	$1 \cdot 10^6$	$1 \cdot 10^4$	$4 \cdot 10^5$	$3 \cdot 10^3$	$1 \cdot 10^6$
Total number of operations	$1 \cdot 10^5$	$2 \cdot 10^6$	$4 \cdot 10^5$	$8 \cdot 10^5$	$1 \cdot 10^5$	$1 \cdot 10^6$
Activation function calls	$1 \cdot 10^3$	$8 \cdot 10^4$	$6 \cdot 10^2$	$2 \cdot 10^4$	$1 \cdot 10^3$	$5 \cdot 10^4$
Size	$4 \cdot 10^4$	$9 \cdot 10^4$	$2 \cdot 10^4$	$6 \cdot 10^4$	$6 \cdot 10^4$	$7 \cdot 10^4$

7 Discussion

One interesting aspect of the problem solution is how good the choice of the hand gestures was. To use gestures as means of communication with a device, the feature data needs to be easily distinguishable and insensitive to performance variations between different people. The separability of the classes looks very promising for the spectrogram features. The analysis on Dataset 1 and 2 showed great potential of informative structure in the features, both the sparse and the IQ spectrograms. The k-nearest neighbour classifier yielded better performance for the IQ data, whereas the convolutional neural networks more successfully classified the sparse spectrograms. It is thereafter of interest to understand how the separability generalized to other people doing the gestures. Looking at Table 11, the accuracy did decrease for all classification algorithms when the data instead was generated by 18 people.

In case of a gesture sensing product, it would be largely beneficial if the user of the product did not have to train the model, but that a pre-trained model could generalize to the new, unseen people. Looking at the results in Tables 12-14, the accuracy decreased for all test sets, which obviously is not so surprising, but rather affirms the problem. The neural networks were suffering the least from this, only by a reduction in performance by a couple of percentage units. However, it was clearly favourable to include more people in the train set, since the accuracy of Model 1, trained on 1 person, was much lower than the models trained on more people, when tested on unseen people.

From the confusion matrices in Figure 18 and 19, one can identify the Button press class as slightly less identifiable than the others, with an average test accuracy of 89.90% when the model was trained on multiple people and not more than 58.41% when the model was trained on only one. From the second confusion matrix, a further observation is however that the three classes Palm hold, Swipe, and Erase were recognized at an incredibly high rate, even by Model 1. These hand gestures therefore demonstrate the best potential for robust classification.

The approach of estimating impulse responses undoubtedly fails to provide as informative feature, which is why it was abandoned at a very early stage. The Elastic net estimations did seem to contain some structure, being recognized 76% of the time. However, a great drawback is the high dimension of these features. From the raw IQ gesture data of 186 sweeps by 228 distance points, the dimension was scarcely reduced to (186,191), which obviously leads to a computationally heavier classification as compared to the other features. When it comes to the wideband integrated dictionary estimations, it manifestly contained no structure. Since the Elastic net estimate version at least seemed to bear some information, it is probable that, in this case, it was the estimation that had failed and not the approach of impulse response features itself. Figure 22 shows an example of how a single impulse response estimate looked.

In contrast to what was done, it could have been a good idea to try classifying the impulse response features, primarily the Elastic net estimations, also using CNNs and not only KNN. CNNs are specifically developed for image classification, also for camera images that can be highly dimensional. By contrast, KNN suffers from the *curse of dimensionality* [18] and could perhaps have performed poorly simply by this fact.

When it comes to the choice of classification algorithms treated, the use of a convolutional neural network is well motivated, thanks to the clear outperformance over the k-Nearest Neighbour classifier. Moreover, among the two options, a 2D CNN classifier outperformed the 1D counterpart. It is however important to note that the way the 1D CNN model was defined could have been unfavourable. Only one size of the kernel was employed, namely $(n, 1)$, which covers a single pixel along the time dimension. The 2D CNN, on the other hand, had kernels of sizes $(3, 3)$ or $(4, 4)$, and these kernels therefore overlap with the picture of several pixels in the time dimension.

On the other hand, such a kernel, covering all the frequency bins, could possibly destroy rather than use the information of frequency over time during the convolutional layers. At each convolution, a certain averaging is performed over the overlap between the image and the kernel. From the resulting value it is impossible to say where in the overlap, or in which frequency bin, the power originates from. This could perhaps have been disadvantageous. The use of many filters, however, offering kernels with different values, could potentially introduce some asymmetries containing information of what frequency bins that contained high power. But this aspect could be a reason to why a 1D convolution did not work as well.

A further consequence of the choice of 1D convolutional kernel, which instead would be favorable, is that the dimensionality was drastically reduced already after the first convolutional layer: from input (m, n) to (m, l) . It should be compared to the shape after one 2D convolution, (m, n, l) . This extra dimension is a plausible reason to the difference in computational complexity, visible in Table 15 and in (11) on page 24. Some other things that in general affects both the 1D and 2D CNNs in complexity are, firstly, how many subsequent convolutional layers there is. The computations necessary for such layers, scales linearly against the number of layers, N . Secondly, the fully connected layers add important computational weight. It is clear from the formulas (10) and (11), that whenever $K > 0$, further arithmetic operations need to be performed. This is not surprising, dense networks are heavy, and big in model size, which was one of the main reasons to consider CNNs and not regular Artificial Neural Networks (ANNs).

7.1 Problems with the Setup

There are some aspects with how the gesture was defined and recorded that was not optimal, especially not when it comes to performing gesture classification in practice. Moreover, the approach to the classification is not flexible to introduction nor removal of classes.

Firstly, the data extraction and labelling had its disadvantages. One problem with the initial setup was that gesture recording started while the hand was installed above the sensor. The argument for that choice was that the movement should be as isolated as possible – to simplify a classification and to make sure that this step worked (with the mind set that a more realistic problem setup could be tried later). From the results, it was clear that several classification algorithms were indeed capable of distinguishing these movements, when they were that separated. However, when it comes to doing classification in real time, any gesture that consists of the hand being above the sensor at the start of the gesture (as for button press) needs to first include the translocation of the hand into the beam. This movement can easily be mixed up

with a swipe movement.

Another issue was that a manual indication of when the gesture was to begin had to be done, which above all caused a timing problem when another person than the one performing the gesture was doing this, i.e., in Dataset 3. The consequence was that some of the features extracted did not actually contain the full gesture. Whether this variation impairs the classification or helps the model generalize more, resulting in a better real-time classifier, is hard to say. In addition, it can have risked that people could not do the gestures as relaxed and naturally as desired.

This approach to supervised learning has a major drawback, namely, that the possible labels are fixed at an initial step. This means, on one hand, that if the network is fed with an example of a so-far unseen class, it will still predict its class among the predefined. On the other hand, the model cannot either be extended to include a new class once it is trained. This has as consequence that if one wishes to introduce a new gesture, the training needs to be redone (perhaps as well as the search of an appropriate architecture).

A final potential problem is that the data set is not extensive. Ideally, one would wish for much more data to fully investigate the ability to use such classification approaches for this task.

8 Conclusions and Future Work

This work adds additional evidence for the potential of gesture recognition using radar based systems, with an average test accuracy of more than 95% for people unseen by the model and 97.6% for data from persons used in the training data. The characterization of gestures by a frequency-time map – the spectrogram – was proved yet again to be a useful tool. One could however do more work on the feature design. For example, one could consider wavelet transforms or Mel-frequency cepstrums, instead of the straightforward STFT. Additional features could be considered, designed with the intention to increase the performance on the more problematic gestures, such as Button press. In cases where there are more specific applications of the gesture sensing in mind, one could investigate the importance of the placing of sensors, and, for instance, how much difference two sensors could increase the performance.

The characterization by filter impulse responses looked less promising for this application. However, having seen successful results in other applications, one cannot discard the idea in itself. It would be useful to understand better why the integrated dictionary estimation of the impulse responses works so poorly in this setting, and to what cases it is limited.

Other tools that would be useful in these kind of projects, would be possibilities to at an early stage analyze separability of classes. In this work, cluster separation measures, such as the Mahalanobis distance and the Bhattacharyya distance could not directly be employed due to singular covariance matrix estimates.

Convolutional neural networks have, as in previous works, proven efficient for the class prediction. It would be interesting to investigate if recurrent neural networks, making use of the time sequence structure, could offer even further improvement in the achievable performance.

The limitation of training data should also be addressed, although at this stage it is quite time consuming to gather huge sets. One way to circumvent this problem could be to use different data augmentation techniques, or even artificially generate features, see [19] for an example.

Appendix A

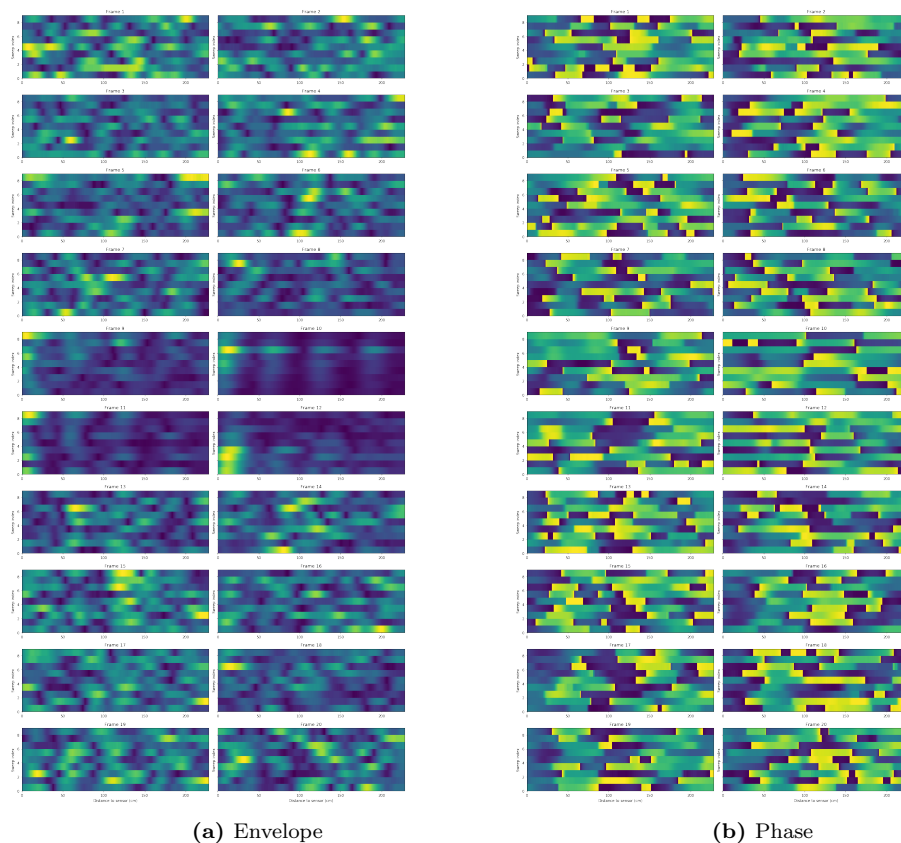
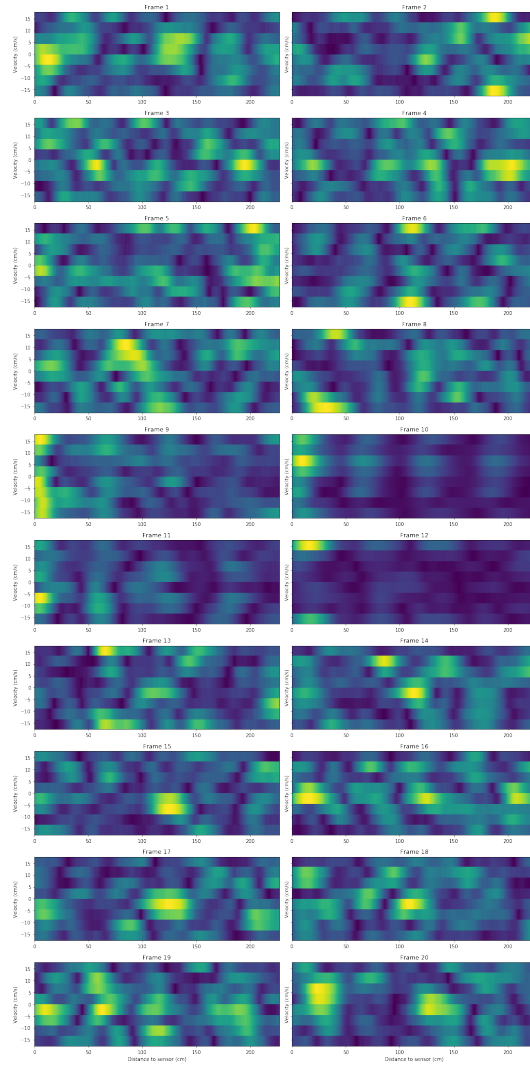
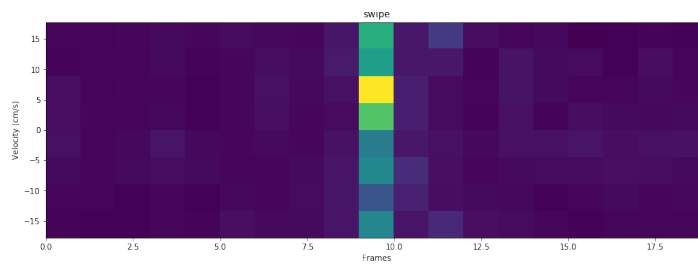


Figure 20: Example of the IQ signal of a gesture (the swipe).



(a) Absolute value of the FFT of the IQ signal in Figure 20.



(b) Spectrogram of an example of the swipe gesture.

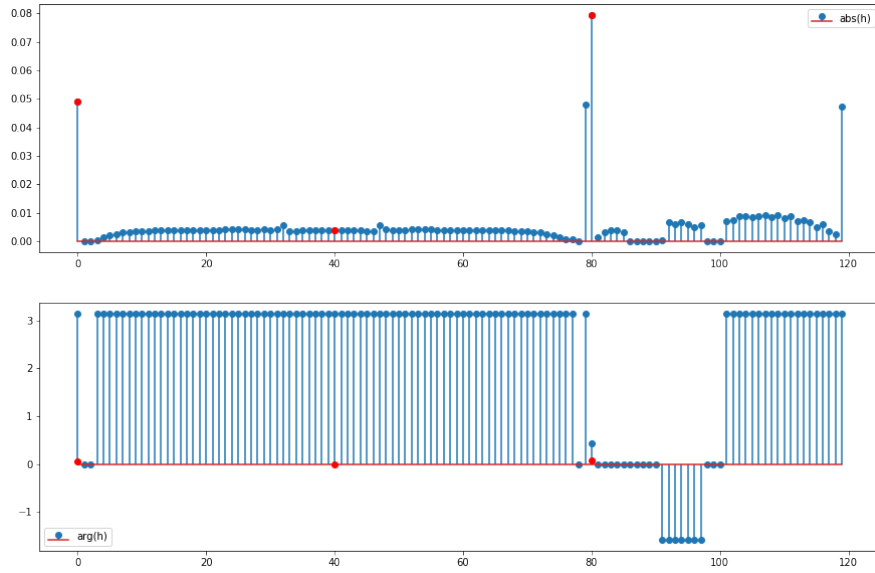


Figure 22: An example of a wideband integrated dictionary impulse response, estimated from a single sweep. The red dots indicates the transition between the active widebands. The sweep is from the same example as in Figure 16a, and the active widebands were 1,2 and 5, out of 6 available.

Appendix B

In this section, the computational complexity of performing a prediction using CNNs is derived in detail. Recall the defined model architecture

$$\text{Input} \longrightarrow \underbrace{[[\text{Conv} + \text{ReLU}] \cdot N \longrightarrow \text{Maxpool}] \cdot M}_{(A)} \longrightarrow \underbrace{[\text{FC} + \text{ReLU} + \text{Dropout}] \cdot K}_{(B)} \longrightarrow \text{FC} + \text{Softmax}, \quad (12)$$

where, moreover, the kernel size, (k_1, k_2) , and the number of filters, l , are hyperparameters of the convolutional layers. Figure 23 illustrates an example architecture of (12).

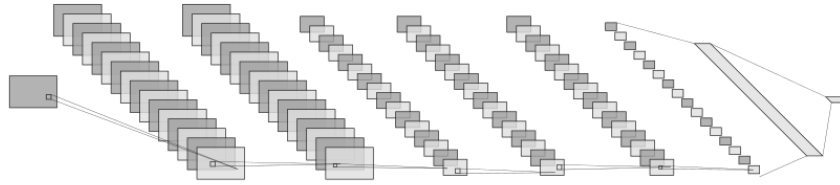


Figure 23: Example architecture, where, $N = 2, M = 2$, and $K = 1$. Further, there are $l = 16$ filters and kernels of size $(k_1, k_2) = (3, 3)$.

When predicting the output using neural networks, a sequence of matrix multiplications and additions are performed, as well as evaluations of non-linear scalar functions. Recall that in a case of a CNN, a kernel of size (k_1, k_2) slides over the picture, and at all positions it is convolved with the current overlap of the picture. The convolution therefore requires $k_1 k_2$ multiplications, followed by $k_1 k_2 - 1$ additions, per position of the kernel. Moreover, a scalar bias is added to the result, yielding, in total, $2k_1 k_2$ operations. The input image is zero-padded to keep the same dimension after the convolution, yielding mn convolutions per filter, or $mn \cdot 2k_1 k_2$ operations per filter, and finally $2mnk_1 k_2 \cdot l$ for the whole layer. In addition, the activation function is called mn times, once for every convolution.

In a maxpooling layer, a downsampling per depth layer is performed in a small region of $(2, 2)$ pixels. This therefore corresponds to finding the maximum value among $2 \cdot 2 = 4$ candidates, for which 3 operations are needed. For each depth layer, $\frac{m}{2} \frac{n}{2}$ downsamplings will occur, assuming for simplicity that m, n are even, yielding a total of $\frac{3}{4} mnl$ operations per full layer.

After a maxpooling layer, the output will be of shape $(\frac{m}{2}, \frac{n}{2}, l)$, which reduces the complexity of the subsequent layers. For a 2D CNN model as defined in (12), the kernel is symmetric, $(k_1, k_2) = (k, k)$, and part (A) gives rise to

$$\sum_{i=1}^M \left(2k^2 \frac{mn}{4^{i-1}} l \cdot N + \frac{3mn}{4^i} l \right) = \sum_{i=1}^M \left(2k^2 N + \frac{3}{4} \right) \frac{mn}{4^{i-1}} l \quad \text{operations,}$$

$$\sum_{i=1}^M N \frac{mnl}{4^{i-1}} \quad \text{calls to activation functions.}$$

Visibly, these numbers depend on both the number of subsequent convolutional layers N and the number of maxpooling layers M .

Part (B) consists of dense and dropout layers. If $K = 1$, there is first a fully connected layer of 100 nodes, to which the flattened stack of images is connected to. That requires a matrix multiplication between $(100, \frac{m}{2^M} \frac{n}{2^M} l) \times (\frac{m}{2^M} \frac{n}{2^M} l, 1)$ and the addition of the bias vector $(100, 1)$, requiring $200 \frac{m}{2^M} \frac{n}{2^M} l$ operations, followed by the call to 100 activation functions. The dropout, at predicting stage, consists of a scaling, i.e., multiplying all the nodes with the dropout probability [23], thus 100 multiplications. The last dense layer, producing the output predictions, contributes with the matrix multiplication $(5, 100) \times (100, 1)$, followed by 5 additions and 5 calls to the softmax function. In total, part (B) therefore requires $200 \frac{m}{2^M} \frac{n}{2^M} l + 1100$ floating point operations and 105 calls to nonlinear activation functions. If, on the other hand, $K = 0$, this number is replaced by the matrix multiplication $(5, \frac{m}{2^M} \frac{n}{2^M} l) \times (\frac{m}{2^M} \frac{n}{2^M} l, 1)$ and the addition of the bias vector of shape $(5, 1)$, which demands $10 \frac{m}{2^M} \frac{n}{2^M} l$ operations and 5 calls to the softmax function.

The full prediction thus requires

$$\underbrace{\sum_{i=1}^M \left(2k^2 N + \frac{3}{4} \right) \frac{mn}{4^{i-1}} l}_{(A)} + \underbrace{\frac{m}{2^M} \frac{n}{2^M} l (10 + K \cdot 190) + K \cdot 1100}_{(B)} \quad \text{operations,} \tag{13}$$

$$\underbrace{\sum_{i=1}^M N \frac{mnl}{4^{i-1}}}_{(A)} + \underbrace{5 + 100K}_{(B)} \quad \text{calls to activation functions,}$$

or

$$\mathcal{O}(k^2 N mnl(1 - 4^{2-M})) + \mathcal{O}(mnl(1 - 4^{2-M})) + \mathcal{O}\left(\frac{mnl}{4^M}(1 + K)\right) + \mathcal{O}(K) \quad \text{operations,}$$

$$\mathcal{O}(N mnl(1 - 4^{3-M})) + \mathcal{O}(K) \quad \text{calls to activation functions.}$$

The model parameters M, N, K, k are finite but visibly introduces computational complexity to the model. If they are not taken into consideration, the model complexity is $\mathcal{O}(mnl)$.

If one considers the case of a 1D CNN, some additional aspects need to be considered. As was discussed when the model was introduced, an initial kernel size of $(n, 1)$ should be used for an image of dimension (n, m) , to convolve along the second axis. In total, the first convolutional layer gives rise to ml convolutions, which requires $2nml$ operations, as well as ml further calls

to the activation function. The subsequent convolutional layers consist, on the other hand, of kernels of size $(l, 1)$, requiring $2ml$ operations (before any downsampling has occurred).

The maxpooling, in this case, is performed on a region of $(2, 1)$, which requires 1 operation per position, in total $\frac{ml}{2}$ operations for the first layer. The shape after all maxpooling layers is therefore reduced to $(l, \frac{m}{2^M})$.

Adding the operations of part (A), there is in total

$$2nml + (N - 1)2ml + \frac{ml}{2} + \sum_{i=2}^M \left(N \frac{2ml}{2^{i-1}} + \frac{ml}{2^i} \right) \text{ operations,}$$

$$\sum_{i=1}^M N \frac{ml}{2^{i-1}} \text{ calls to activation functions.}$$

Thereafter, as before, the processed image is reshaped to a one-dimensional vector, which is given as input to fully connected layers. Either, if $K = 1$, an initial $200 \frac{ml}{2^M}$ operations in the first fully connected layer are required. Calls to activation functions and additional operations for the dropout and the last dense layer are needed, just as before. If $K = 0$, a total amount of $\frac{10ml}{2^M}$ operations plus 5 calls to the softmax functions are performed. Therefore, a full prediction consists of

$$\underbrace{2nml + (N - 1)2ml + \frac{ml}{2} + \sum_{i=2}^M \left(N \frac{2ml}{2^{i-1}} + \frac{2ml}{2^i} \right)}_{(A)} + \underbrace{\frac{ml}{2^M} (10 + 190K) + K \cdot 1100}_{(B)} \text{ operations,}$$

$$\underbrace{\sum_{i=1}^M N \frac{ml}{2^{i-1}}}_{(A)} + \underbrace{5 + 100K}_{(B)} \text{ calls to activation functions,}$$

(14)

or

$$\mathcal{O}(ml(n - 1)) + \mathcal{O}(mlN(1 - 2^{1-M})) + \mathcal{O}(ml(1 - 2^{1-M})(N + 1)) + \mathcal{O}\left(\frac{ml}{2^M}(1 + K)\right) + \mathcal{O}(K) \text{ operations,}$$

$$\mathcal{O}(Nml(1 - 2^{2-M})) + \mathcal{O}(K) \text{ calls to activation functions.}$$

References

- [1] J. Lien, N. Gillian, M. E. Karagozler, P. Amihoud, C. Schwesig, E. Olson, H. Raja, and I. Poupyrev. Soli: Ubiquitous gesture sensing with millimeter wave radar. *ACM Trans. Graph.*, 35(4):142:1–142:19, Jul 2016.
- [2] A. Hassaniien and H. Braham. *Fundamentals of Indoor Radar*. CRC Press, Boca Raton, US, 2018. In Amin, M. G. (Ed.), *Radar for Indoor Monitoring* (pp. 1-20).
- [3] R Watson-Watt. Radar in war and in peace. *Nature*, 156:319–324, 1945.
- [4] T. K. Sarkar and M. Salazar Palma. A history of the evolution of radar. *2014 44th European Microwave Conference*, pages 734–737, Oct 2014.
- [5] S. Azevedo and T. McEwan. Micropower impulse radar. *IEEE Potentials*, 16:15–20, Apr-May 1997.
- [6] H.-C. Kuo, C.-C. Lin, C.-H. Yu, P.-H. Lo, J.-Y. Lyu, C.-C. Chou, and H.-R. Chuang. *IEEE Transactions on Microwave Theory and Techniques*, 64:1018–1028, Apr 2016.
- [7] B. Borja, J. A. Tirado, and Jardon H. An overview of uwb antennas for microwave imaging systems for cancer detection purposes. *Progress In Electromagnetics Research B*, 80:173–198, May 2018.
- [8] M. G. Amin. *Radar for Indoor Monitoring (Preface)*. CRC Press, Boca Raton, US, 2018.
- [9] B. Barbelo. (don’t) hold the phone: new features coming to pixel 4. *Google*, Jul 2019. Accessed at <https://www.blog.google/products/pixel/new-features-pixel4/> Nov 16 2019.
- [10] H. Dahlberg and A. Evertsson. Short range gesture sensing and classification using pulsed millimeter-wave radar and convolutional neural networks, 2019. Master’s Thesis.
- [11] S. Heunisch. *Millimeter-Wave Radar for Low-Power Applications*. PhD thesis, Lund University, Aug 2019.
- [12] Jonas Erb. Embedded gesture recognition using novel mini-radar sensors, 2018. Master’s Thesis.
- [13] Acconeer AB. An illustration of how the sparse service samples reflected waves from a moving object., 2019. Accessed at <https://acconeer-python-exploration.readthedocs.io/en/latest/services/sparse.html> Nov 19 2019.
- [14] Acconeer AB. Illustration of envelope and phase change of a received pulse for a reflection from a moving object, what is returned from the iq service is in cartesian form., 2019. Accessed at https://acconeer-python-exploration.readthedocs.io/en/latest/sensor_introduction.html Nov 19 2019.
- [15] R. G. Lyons. *Understanding Digital Signal Processing (3rd edition)*. Prentice Hall, 2011.
- [16] Acconeer AB. An illustration of the sparse data frames consisting of a number of sweeps., 2019. Accessed at <https://acconeer-python-exploration.readthedocs.io/en/latest/services/sparse.html> Nov 21 2019.

- [17] S. Cang, J. Swärd, X. Sheng, and A. Jakobsson. Toeplitz-based blind deconvolution of underwater acoustic channels using wideband integrated dictionaries. Submitted, 2019.
- [18] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009.
- [19] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. Accessed at <http://www.deeplearningbook.org>.
- [20] P. L. Pröve. 2d convolution using a kernel size of 3, stride of 1 and padding, 2017. Accessed at <https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d> Nov 25 2019.
- [21] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *American Mathematical Society*, 19(90):297–301, Apr 1965.
- [22] S. Arya, D. M. Mount, N. S. Netanyahu, Wu Silverman, R., and A. Y. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, Nov 1998.
- [23] N. Srivasta, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, Jun 2014.