Universidad Complutense de Madrid

Real-time gesture recognition with new-generation smart materials

Facultad de Ciencias Matemáticas

Authors: Alberto Alarcón
Adrian Blázquez
Manuel Martín-Mora
Paula Urruchi
Omayra Yago

Coordinator: Ivan Tyukin, University of Leicester, UK

Assistant: Carmen García-Miguel

June 11, 2021

Abstract

The main purpose of the Modelling Week is to promote the use of mathematical methods and models in research, industry, innovation, and management in the knowledge economy. Master students and participants work in small groups on real industrial problems proposed by companies under supervision of several qualified instructors. The 15th Modelling Week was organised within the Master in Mathematical Engineering program at the Faculty of Mathematics at the Complutense University of Madrid (UCM) in collaboration with the Institute of Interdisciplinary Mathematics (IMI). This special event was held in an online format from June 7 to June 11, 2021.

The 15th Modelling Week's first problem was proposed by Leicester University, together with Tangi0 Ltd. The principal goal of this problem is to develop and implement recognition algorithms using a newly-patented Etee device manufactured and produced by Tangi0 Ltd, which is a patented finger-sensing technology that allows gamers and enterprise users to control electronic interactions using touch and pressure.

Objectives and Work Plan

Objectives

The main purpose of the problem is to create a learning model such as the one explain previously. To achieve these we have encouraged ourselves to achieve the following goals:

- Understand the data provided and perform a transformation to said data for the utilisation of learning techniques.
- Construct a learning model through the utilisation of neural networks.
- Analyse the results and draw conclusion on the performance of the said models.

Work Plan

1. Undertanding of the problem and the device we use in it.
2. Study of theoretical background needed for the problem.
3. Creating of the frequency diagrams and spectrograms with the first data given.
4. Test neural networks with both datasets of images.
5. Incorporation of new data and retrain the neural network with the corresponding new images.
6. Study of the results.

Contents

Chapter 1

Introduction

In this chapter, we will introduce the device used in the collection of the data and the problem we must resolved with said data. Furthermore, we will give introduce several notions on Learning theory as theoretical background on classifiers.

1.1 Etee Device

The device (pictured below) is a patented finger-sensing technology enabling gamers and enterprise users to control electronic interactions via the power of touch and pressure, through the use of 5 sensors. Instead of traditional controllers with mechanical buttons the device features a smart control surface reacting to brush and touch.

Figure 1.1: Etee device

This interesting item is developed by the British company TG0. This start-up company is specialised in the development of objects with a tactile surface. Some examples of these products are the ones incorporated in the interior of the car, in some computer peripherals as Ette or in some sports items and medical devices.

1.2 Problem

The main purpose of the problem is to develop a machine learning model, capable of recognising the movements of the Etee user. This can be used in the following examples,

where the users are playing videogames. The model that is expected to be develop has to recognise each user movement and reproduce it exactly.


We will primarily focus on three movements: a grip which we will call a customise movement, a palm movement and a soft touch movement, which consist of touching the device with our index finger.

1.3 Learning Theory

In this section, we will make several assumptions and introduce several notions to verify the risk and accuracy performance of the futures models we will be executing.

Assumption 1.3.1 (Data) *We will assume the following:*

- *Let X be a set of "measurements" , "features", etc.*
- *Let L be a set of "labels".*
- *Let P be a probability distribution defined in $X \times L$.*

*A **data sample of size** m is $((x_1, y_1), (x_2, y_2), \dots (x_m, y_m))$, m pairs that are identically and independently drawn from the distribution P.*

Remark 1.3.1 *Note that the distribution P is not known. Crucially, however, we assume that it exists, and that the data is independently drawn from it.*

Assumption 1.3.2 (Classifier) *A **classifier** is a map (a function)*

$$f : X \times W \to L,$$

where $W \subset \mathbb{R}^d$ is a set of parameters.
For a given $x_i \in X$ and $w \in W$ the classifier generates a "predicted" label

$$\hat{y}_i(w) = f(x_i, w). \tag{1.1}$$

Remark 1.3.2 *Some part of the theory we are going to discuss is not limited to classifiers.*

Now we will introduce several notions related to Risk theory.

Definition 1.3.1 *The **Loss funtion***

$$Q : L \times L \to \mathbb{R}_{\geq 0} \tag{1.2}$$

is a tool to measure how well predicted labels match those generated by the data distribution (recall that both x_i, y_i are from P). For a classification (binary) problem,

$$Q(\hat{y}, y) = \begin{cases} 0, & \hat{y} = y \\ 1, & \hat{y} \neq y. \end{cases} \tag{1.3}$$

Definition 1.3.2 *The **Risk functional** is the expected loss*

$$R(w) = E_{x,y}(Q(f(x, w), y)). \tag{1.4}$$

Assumption 1.3.3 *$R(w)$ exists $\forall\ w \in W$.*
We can further calculate in a theoretical manner the value of such functional.

$$R(w) = \int Q(f(x, w), y) \cdot p(x, y) \cdot dxdy$$

Problem 1.3.1 ***Core problem**: Now we reach certain problem, we cannot compute it, since $p(x, y)$ is unknown.*

So, we will try to approximate that value through the following definition:

Definition 1.3.3 (Empirical Risk Functional)

$$R_{emp}(w) = \frac{1}{m} \cdot \sum_{i=1}^{m} Q(f(x_i, w), y_i)$$

As the name implies this risk functional is empirical meaning we can evaluate it. But, we do not know how well is the risk functional represented by the empirical one.

Ergo, we ask ourselves the following question. "Given w and m (size of the sample), how far is $R(w)$ from $R_{emp}(w)$?"

Let us put this into perspective. Suposse that we have a "learning algorithm" which, for the given sample of m training data (independently drawn from P) generates w_m. We may be happy with the value of $R_{emp}(w_m)$. We want to know how far $R_{emp}(w_m)$ ios from the unknown true Expectation, $R(w_m)$. In particular, if $\mathbb{P}$ is the probability, we wanr to know

$$\mathbb{P}(R(w_m) - R_{emp}(w_m) < \epsilon),$$

that is, $R(w_m) \leq R_{emp}(w_m) + \epsilon - a$ risk bound we want to know. Let us suppose that we

Step 1: picked $w \in W$.

Step 2: generated (independently on w) an independently sample of m pairs (x_i, y_i) and

Step 3: evaluated $R_{emp}(w)$. Let us now estimate $R(w) - R_{emp}(w)$. Recall that (in our settings) $Q(f(x_i, y_i), y_i)$ is a random variable taking values in $\{0, 1\}$. Let us denote

$$\begin{aligned} Q_i &= Q(f(x_i, w), y_i) \\ R_{emp} &= \frac{1}{m}\sum_{i=1}^{m} Q_i \\ R(w) &= E(Q) \end{aligned}$$

and we have the following inequality (Hoffding inequality)

$$\mathbb{P}(|R(w) - R_{emp}(w)| < \epsilon) = 1 - \mathbb{P}(|R(w) - R_{emp}(w)| \geq \epsilon) =$$
$$\mathbb{P}(|R(w) - R_{emp}(w)| \geq \epsilon) = \mathbb{P}\bigg(E(Q) - \frac{1}{m}\sum_{i=1}^{m} Q_i| \geq \epsilon\bigg) \leq 2e^{-2m\epsilon^2}$$

One-sided version is

$$\mathbb{P}\bigg(\frac{1}{m}\sum_{i=1}^{m} Q_i - E(Q) \geq \epsilon\bigg) \leq e^{-2m\epsilon^2}. \tag{1.5}$$

If we denote $\delta = 2e^{-2m\epsilon^2}$, then

$$\begin{aligned} \ln(\delta) &= \ln(2) - 2m\epsilon^2 \\ m\epsilon^2 &= \frac{1}{2}\bigg(\ln(2) - \ln(\delta)\bigg) \\ \epsilon^2 &= \frac{1}{m}\frac{1}{2}\bigg(\ln\left(\frac{2}{\delta}\right)\bigg) \\ \epsilon &= \sqrt{\frac{1}{2m}\ln\left(\frac{2}{\delta}\right)} \end{aligned}$$

Hence

$$\begin{aligned} &\mathbb{P}\bigg(|\frac{1}{m}\sum_{i=1}^{m} Q_i - E(Q)| < \epsilon\bigg) = \\ &1 - \mathbb{P}\bigg(|\frac{1}{m}\sum_{i=1}^{m} Q_i - E(Q)| \geq \epsilon\bigg) = \\ &1 - \mathbb{P}\bigg(|\frac{1}{m}\sum_{i=1}^{m} Q_i - E(Q)| \geq \sqrt{\frac{1}{2m}\ln\left(\frac{2}{\delta}\right)}\bigg) \\ &\geq 1 - \delta \end{aligned}$$

with probability $1 - \delta$,

$$|R_{emp}(w) - R(w)| \leq \sqrt{\frac{1}{2m}\ln\left(\frac{2}{\delta}\right)}$$
$$R(w) \leq R_{emp}(w) + \sqrt{\frac{1}{2m}\ln\left(\frac{2}{\delta}\right)}$$

Isn't this nice? Well, the problem is that w was chosen independently on data. In our work, we selct w on the basis of data (use the data to select w). Let w_m be the minimizer of R_{emp}

$$w_m = argmin_{w \in W} R_{emp}(w),$$

and let the set W be finite, $|W| = N$. One way to produce an estimate for $R(w_m)$ is to consider

$$\begin{aligned}
&\mathbb{P}\left(sup_{w \in W} |R(w) - R_{emp}(w)| < \epsilon \right) \\
&\mathbb{P}\left(sup_{w \in W} |R(w) - R_{emp(w)}| < \epsilon \right) \leq \sum_{j=1}^{N} \mathbb{P}\left(|R(w_j) - R_{emp}(w_j)| < \epsilon \right) \\
&\leq \text{ (from previous derivations)} 2Ne^{-2m\epsilon^2}.
\end{aligned}$$

Denoting

$$\begin{aligned}
2Ne^{-2m\epsilon^2} &= \delta \\
\ln(2N) - 2m\epsilon^2 &= \ln(\delta) \\
\epsilon &= \sqrt{\frac{1}{2m} \ln\left(\frac{2N}{\delta}\right)} \\
R(w_m) &\leq R_{emp}(w_m) + \sqrt{\frac{1}{2m} \ln\left(\frac{2N}{\delta}\right)}
\end{aligned}$$

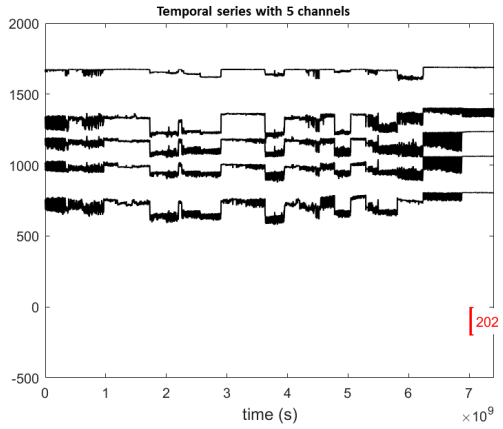
Chapter 2

Creation of datasets

2.1 Introduction of data

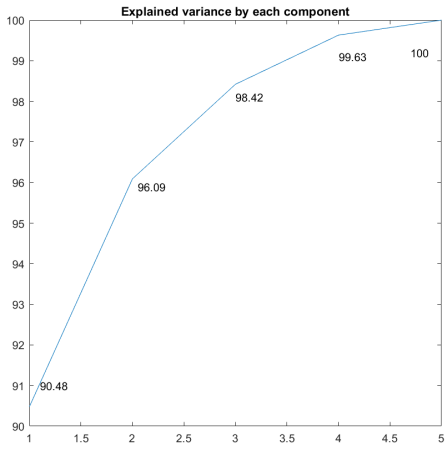
We were provided with 2 batches of excel files with information on each movement. The first batch, consisting of seven files for each movement, will be used for both training and testing. And the last batch, consisting of only one excel file for each movement, will be only used for testing.

As we can see in the image below, when performing the different movements to the device, which have 5 sensors as we mentioned earlier, each of these produces a wave of unique frequency for each person and time.



When seeing the different representations of the different data provided we found that the last 4 signals had similar trajectories, which made us think that, when working with the data, we would not need the 5 signals. So we decided to do a principal component analysis.

The first analysis was carried out with the first batch of data obtained, and, as we can see, almost all the variability of the time series is explained by the first sensor, with more than 90%. Even so, we take the first three main components to carry out the subsequent studies since, as we see in the graph of the accumulated explained variability, these come to explain more than 98% of the variability of the data.

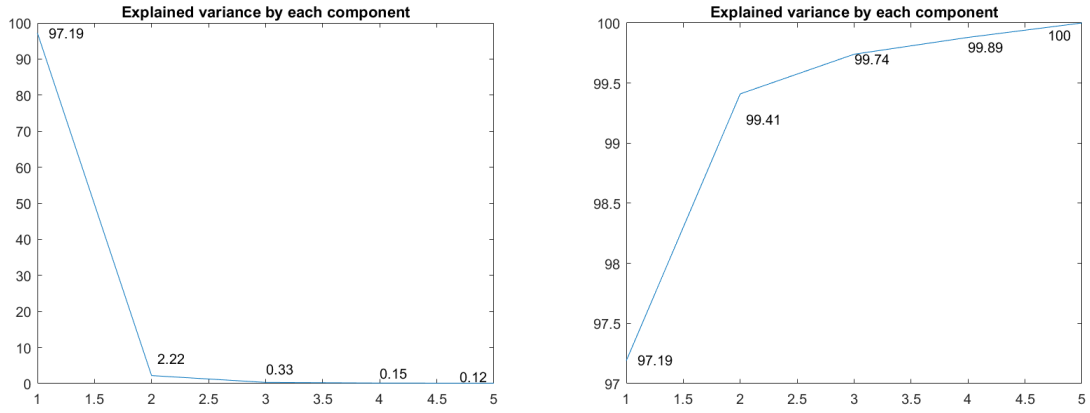


In addition, we carried out a study of the correlations between the different signals to check if it was a good decision to reduce the signals to work. As we found, all the signals have a high correlation between them, the lowest being 0.6058 between the first and fifth sensor and the highest 0.9316 between the second and 5. So with this we confirm that it is correct to reduce the dimension of the problem to work with 3 sensors.

	1	2	3	4	5
1	1.0000	0.6266	0.7565	0.7556	0.6058
2	0.6266	1.0000	0.8787	0.8123	0.9316
3	0.7565	0.8787	1.0000	0.9640	0.8865
4	0.7556	0.8123	0.9640	1.0000	0.8733
5	0.6058	0.9316	0.8865	0.8733	1.0000

After producing the model for the first batch, which will be explained in the following chapters, our coordinator Ivan emailed us the second batch of data, for us to join with the previous batch in order to perform a more in depth study.
Just like with the first batch we will carried out a principal component analysis to try reducing the number of channels employed in the later studies.

We obtained the following representations:



(a) Explained Variance

(b) Cumulative Variance

Figure 2.1: Caption

In the first one we can see the explained variability for each channel/sensor, and in the second one, the cumulative variability. Now, we can see how in this case the first sensor has more importance than with the previous analysis, reaching a 97% of the variability of the sample. If we were studying this batch separately we will only choose retain the first sensor for the following studies. However, as our aim was to join both batches and see how the results vary when introducing a new data source, we will retain, as previously, the first three channels.

We will also perform a correlation analysis between the different sensors with this batch of data, we can see how the fact, that most information given is redundant, is confirmed again because as we can see in the correlation matrix the smallest correlation between different sensors is 0.9160. The outtake we get from this is that we will only need a sensor in order to achieve quality results, since the rest of the information seems to be not relevant in the study.

	1	2	3	4	5
1	1.0000	0.964	0.9580	0.9160	0.9236
2	0.9645	1.0000	0.9753	0.9265	0.9441
3	0.9580	0.9753	1.0000	0.9716	0.9778
4	0.9160	0.9265	0.9716	1.0000	0.9876
5	0.9236	0.9441	0.9778	0.9876	1.0000

Table 2.1: Second batch correlation matrix between the 5 sensors

2.2 Formation of datasets

As we have previously stated we are working with information about three sensors. We thought the best approach to the resolution of the problem with such data was through the creation of images and performing a neural network using that new generated data.

We decided to focus in creating two types of datasets:

- The first one is representing the data as frequencies diagrams.
- The second one is representing the data as spectograms.

In both cases, we decided to use a 300 entries window to create the new images.

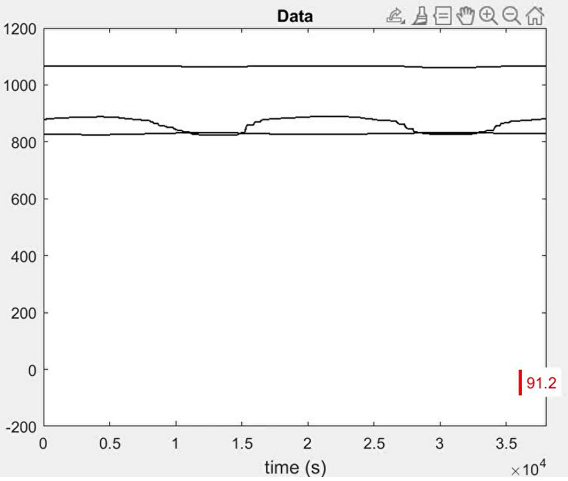


Figure 2.2: A 300 window representation of the data

Furthermore, we took an 80% sample for the training data and a 20% sample for the testing data. But, in order to avoid any type of correlation between the samples as we are taking a 300 entries window, we have to "create" a transition/frontier between both samples. Graphically this means

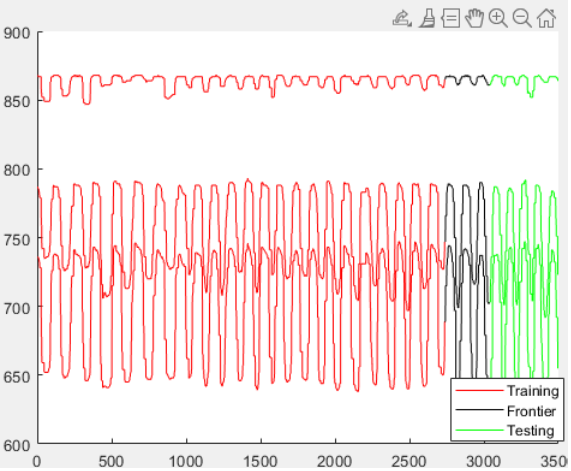


Figure 2.3: A representation of the training, frontier and testing samples

Moreover, in order to reduce the space usage in our computer we decided to create the images with size 32 by 32.

2.2.1 Frequency Diagrams

Now, we will show the code used to create the first dataset of images, the frequency diagrams.

```
str = [...
    './ML_data/etee_customise_1.csv';
    './ML_data/etee_customise_2.csv';
    './ML_data/etee_customise_3.csv';
    './ML_data/etee_customise_4.csv';
    './ML_data/etee_customise_5.csv';
    './ML_data/etee_customise_6.csv';
    './ML_data/etee_customise_7.csv';
    './ML_data/etee_customise_8.csv'];
it_train = 1; it_test = 1;
for k = 1:8
    var1 = csvread(str(k,:));
    w = 300;
    i = 2;
    while i+w < 0.8*size(var1,1)
        Fs=1/(var1(end,1)-var1(2,1));
        LFP=var1(i:w+i,2:4)';
        PlotEEG(LFP,Fs)
        saveas(gcf,['./Train/Customise/' num2str(it_train) '.png'])
        aux = imread(['./Train/Customise/' num2str(it_train) '.png']);
        aux = imresize(aux,[32,32]);
        imwrite(aux,['./Train/Customise/' num2str(it_train) '.png'])
        i=i+1; it_train=it_train+1;
    end
    i = floor(0.8*size(var1));
    while i+w < size(var1,1)
        Fs=1/(var1(end,1)-var1(2,1));
        LFP=var1(i:w+i,2:4)';
        PlotEEG(LFP,Fs)
        saveas(gcf,['./Test/Customise/' num2str(it_test) '.png'])
        aux = imread(['./Test/Customise/' num2str(it_test) '.png']);
        aux = imresize(aux,[32,32]);
        imwrite(aux,['./Test/Customise/' num2str(it_test) '.png'])
        i=i+1; it_test=it_test+1;
    end
end
```

Figure 2.4: Code employed to generate the frequency diagrams

The code shown is used to create the images for the customise movement, the other movement's images are created in a similar way modifying the input and output routes. As we said, we have to divide the dataset into training and testing sets. To do that, and in order to take the same percent of information from each file, we take the 80% of each of the 8 files to form the training set, that includes the frontier, and the 20% for testing. We also have to resize all the images into 32 times 32 size.

We must give importance to the output folders generated for each movement, for they are fundamental in the properly implementation of the neural network as we are solving a classification problem.

2.2.2 Spectograms

Following the previous structure we will shown the code used to create the second dataset of images, the spectograms.

```
var1=csvread('Test_palm.csv');
LFP=var1(2:end,2:end);

for i = 500:1000
    X=LFP(i:i+299,1);
    Y=LFP(i:i+299,2);
    Z=LFP(i:i+299,3);
```

```
8       f1=figure(1);
9       spectrogram(X,'yaxis');
10      axis off
11      colorbar('off')
12      saveas(f1, ['./canal1/sptest_palm' num2str(i) '.png']);
13      f1=figure(1);
14      spectrogram(Y,'yaxis');
15      axis off
16      colorbar('off')
17      saveas(f1, ['./canal2/sptest_palm' num2str(i) '.png']);
18      f1=figure(1);
19      spectrogram(Z,'yaxis');
20      axis off
21      colorbar('off')
22      saveas(f1, ['./canal3/sptest_palm' num2str(i) '.png']);
23      saveas(f1, ['./RGB/test/palm/sptest_palm' num2str(i) '.png']);
24      saveas(f1,['./RGB/test/palm/resize/sptest_palm' num2str(i) '.
            png']);
25
26      l1=imread(['./canal1/sptest_palm' num2str(i) '.png']);
27      l2=imread(['./canal2/sptest_palm' num2str(i) '.png']);
28      l3=imread(['./canal3/sptest_palm' num2str(i) '.png']);
29      mix1=imfuse(l1,l2);
30      fin=imfuse(mix1,l3,'ColorChannels',[1 1 2]);
31      imwrite(fin,['./RGB/test/palm/sptest_palm' num2str(i) '.png']);
32      aux = imresize(fin,[32,32]);
33      imwrite(aux,['./RGB/test/palm/resize/sptest_palm' num2str(i) '.
            png']);
34  end
```

The essence of the above code is to generate a spectogram for each channel and superimpose them in order to generate the spectogram picture we will be working with. In addition, we must take into account we have reduce the sample size for the spectogram case due to the limitation in time and space.
Finally, as we have previously mention this code only generates the images for one gesture, in order to generate the others we have to change the input and output routes.

Now we will see images of a spectogram for one channel and a superimposed spectogram.

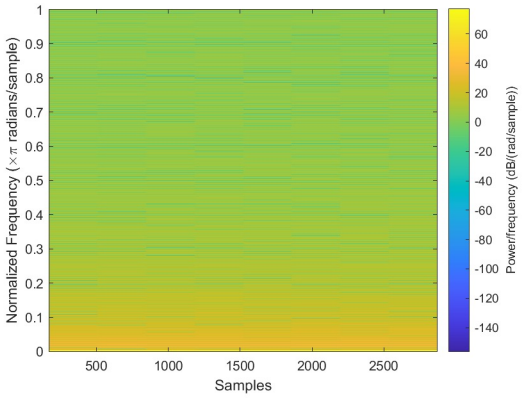


Figure 2.5: Picture of one spectogram

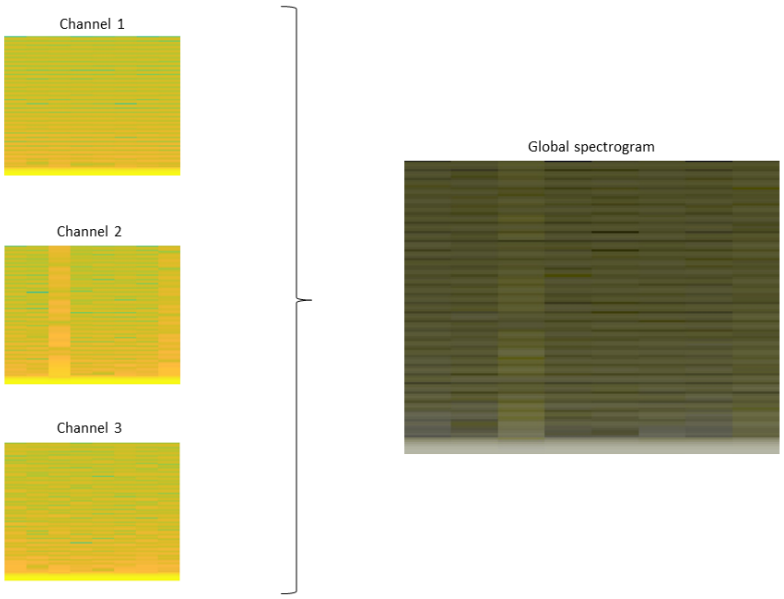


Figure 2.6: Pictures of the initial and the superimposed spectogram

The colour that appears here change depending of the greater frequency of colour in each channel as you can see in the previous picture, being channel 1 the one for red, 2 for green and 3 for blue. Then we mix all the channel and we obtain the big image. In this image, if we get grey colours is because the channels have similar intensities.

Chapter 3

Frequency Diagrams

In this chapter, we created a neural network that classifies the images with the raw data the device produces.

3.1 Pre-processing of data

The first thing we did before starting working on the architecture of our network was make sure the train samples were balanced, they were not, while palm and soft_touch gestures had between 17 and 18 thousand images, the gesture customized had around 10 thousand more pictures, to balance this train set we deleted from each file a small portion of the last records till we balanced this train dataset.

After doing this, we uploaded them into matlab with the following code:

```
TrainData = imageDatastore('./gestos/train','IncludeSubfolders',...
            true,'LabelSource', 'foldernames');
TestData = imageDatastore('./gestos/test','IncludeSubfolders',...
            true,'LabelSource', 'foldernames');
```

By doing it this way, we automatically have the correct label stored for every image since we have all the data organised in folders.

3.2 Neural Network

The architecture of the network follows an architecture like the one we can see here:

It is formed by an input layer, a convolution and a relu layer, a pooling layer, a fully connected layer formed by 2 hundred and 12 neurons, a fully connected layer with the same neuron as classes, in this case 3, a softmax layer and lastly the classification layer.

Also, the method used for the training is the stochastic gradient descent with momentum optimizer, with an initial learning rate of 0.0005 and maximum of 1 full pass of the data.

3.3 Training & Testing

We trained the network with the train dataset, the layers created and the options mentioned in the previous section. With an input of approximately 53 thousand pictures

```
layers = [imageInputLayer([32,32,3]);convolution2dLayer(3,32);
    reluLayer; maxPooling2dLayer(2,'Stride',2);fullyConnectedLayer(212);
    fullyConnectedLayer(nClsTrain); softmaxLayer; classificationLayer];

options = trainingOptions('sgdm','InitialLearnRate', 0.0005,...
    'MaxEpochs',1);
```

it takes 5 minutes to train, then with this trained net, we classified the train and test data and created the confusion matrix for each one.

```
tic

net = trainNetwork(TrainData, layers, options);

toc
```

```
labelTrain = classify(net, TrainData);
Accuracy = mean(labelTrain == TrainData.Labels);
disp(['Classification accuracy (Train) = ' num2str(100*Accuracy,3) '%'])

confusionmat(labelTrain, TrainData.Labels)

labelTest = classify(net, TestData);
Accuracy = mean(labelTest == TestData.Labels);
disp(['Classification accuracy (Test) = ' num2str(100*Accuracy,3) '%'])

confusionmat(labelTest, TestData.Labels)
```

3.3.1 Train dataset

	Customise	Palm	Soft-Touch
Customise	18085	0	0
Palm	0	18077	0
Soft-Touch	0	0	17098

As we can see in the confusion matrix, with just 5 minutes it classifies perfectly all 53 thousand images.

3.3.2 Test datset

	Customise	Palm	Soft-Touch
Customise	4131	0	0
Palm	0	4697	0
Soft-Touch	0	348	4257

As for the test data, it classifies correctly 97.4 % of them in just 1 minute, almost 13 thousand new different gestures from different people classified correctly.

3.4 Testing with New Data

But, a day after we were given new data, we ran it in the network and it just classified 28.6% of it correctly, setting as a default label the gesture customize.

	Customise	Palm	Soft-Touch
Customise	1422	1905	1905
Palm	0	0	0
Soft-Touch	0	0	19

To solve this problem we tried two different paths:

Firstly, we thought this could be an overfitting problem because we trained the net with so many images, so we added two dropout layers and we didn't obtain any improvement by doing this.

But after, we changed the first 35% pictures into the training set and retrained it again, and now it correctly classifies 89.7% of the images, taking 8 minutes to retrain the net and just 24 seconds in classifying the new inputs.

	Customise	Palm	Soft-Touch
Customise	1422	0	411
Palm	0	1233	0
Soft-Touch	0	0	1008

We ran this one more time adding less data to the training set, just 20% of the data, and by doing this we got 92% of the test data classificated correctly in just 30 seconds.

	Customise	Palm	Soft-Touch
Customise	1802	0	328
Palm	0	1053	0
Soft-Touch	0	0	911

This means that for an optimal use of this network it will have to be retraining constantly, but this is not impossible, companies like apple and their finger recognition device in its phones have this kind of technology that constantly retrains. This is not ideal, but it is a good start, since no one has done this before for gesture recognition.

Chapter 4

Spectograms

4.1 Neural Network

First, we use a neural network with the following structure:

Figure 4.1: Initial CNN

First network doesn't have too many layers, a first layer (input layer, with 32x32x3 images), a second convolutional layer with 32 filters with a filter size of 3, a relulayer layer, a 2x2 maxpooling layer with Stride 2, a dense layer with 512 neurons and the layer with 3 neurons, ending with a softmax layer and the classification layer. We use a learning rate of 0.0002, and 20 epochs.

But when testing with different datasets, since we see overfitting at first, we will improve the network obtaining a final structure:

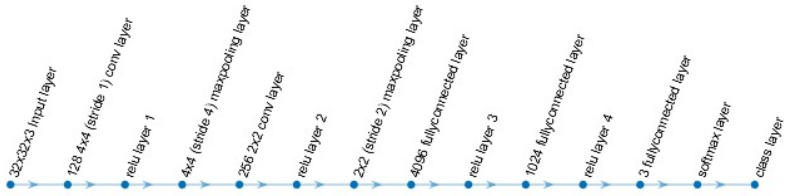


Figure 4.2: Final CNN

Input layer (32x32x3); convolutional layer with 128 filters, size 4x4 and stride 1; relulayer; maxpooling layer 4x4 with stride 4; convolutional layer with 256 filters size 2x2 and stride 1; relulayer; maxpooling layer 2x2 with stride 2; fullyconnected layer with

4096 neurons; relulayer; fullyconnected layer with 1024 relulayer neurons; relulayer; fullyconnected layer with 3 neurons; softmax layer; classification layer.

4.2 Training & Testing

In the first case, we will have a dataset of 1000 data per gesture. 80% trainset and 20% test set. In this case we obtain an accuracy of 97% and the network takes 5 seconds. Since it is data that comes from the same sample, this very good result may be due to overfitting.
Looking at the confusion matrix, it is seen that there are very few errors, where rows are predictions and columns are true result.

	Customise	Palm	Soft-Touch
Customise	199	16	0
Palm	0	185	0
Soft-Touch	2	0	201

As before to evaluate the risk, with probability $1 - \delta$ (where we choose different delta) we will calculate epsilon. We try to estimate R (W), we calculate delta and epsilon. So, we obtain the following results, where we estimate that R will be Risk empirical functional + epsilon for probability $1 - \delta$.

Epsilon result:

```
Risk empirical functional = 0%
With delta = 0.1, we obtain epsilon = 0.158
With delta = 0.05, we obtain epsilon = 0.17533
With delta = 0.01, we obtain epsilon = 0.21013
With delta = 0.001, we obtain epsilon = 0.25168
```

Figure 4.3: Empirical Risk for original Dataset

4.3 Testing with New Data

In the second case, with the data from the previous train, we train the same network for new data and obtain results close to 52-53%, which, without being the worst, can be improved. Therefore, we try to improve the network by adding layers and changing parameters (filtersize, numfilters). With this network we improve the results up to 63.8% in 37 seconds. This result is better in this case than the one we obtained with the network in this case from the images of the 28.6% signals. In the confusion matrix customise gesture and palm gesture is well evaluate, however, soft touch has many more errors.

	Customise	Palm	Soft-Touch
Customise	884	341	607
Palm	0	659	21
Soft-Touch	117	0	372

Epsilon result:

```
Risk empirical functional = 0.37333%
With delta = 0.1, we obtain epsilon = 0.07066
With delta = 0.05, we obtain epsilon = 0.07841
With delta = 0.01, we obtain epsilon = 0.093971
With delta = 0.001, we obtain epsilon = 0.11255
```

Figure 4.4: Empirical Risk for new Dataset

We tested a third case, adding a small sample to the existing training set of 300 per gesture from this last dataset, and we retest the network. In this case, after several executions, we obtain a 66.8% accuracy, better than 63.8%. We observe that somewhat better results are obtained by adding a small number of data from the new dataset. Also, looking at the confusion matrix, we see that soft touch accuracy has been improved quite a bit.

	Customise	Palm	Soft-Touch
Customise	699	241	466
Palm	0	759	4
Soft-Touch	302	0	530

Epsilon result:

```
Risk empirical functional = 0.28%
With delta = 0.1, we obtain epsilon = 0.07066
With delta = 0.05, we obtain epsilon = 0.07841
With delta = 0.01, we obtain epsilon = 0.093971
With delta = 0.001, we obtain epsilon = 0.11255
```

Figure 4.5: Empirical Risk for new training dataset

Chapter 5

Conclusions

In this work, we have applied neural networks with different sets of images. First, we apply these neural networks on signals, taking an average training time of 6 minutes. It doesn't need much computing memory since it uses directly the image that corresponds with the frequency of a specific signal. This neural network application is the best option in a more personalized use.

In the other hand, we apply neural networks with a set of spectrograms generated with the different signal frequencies we have. This network works only with one training operation, taking seconds to do it. This second method is more robust but, more computationally expensive, because is necessary to calculate previously all the spectrograms we are working with.

The two setups have their own pros and cons and the choice between the two isreally dependent on what the use case is. One requires more preprocessing than the other, they are both suitable for customized learning "at home" by a user.

Chapter 6

Appendix

6.1 Neural network for spectrogram generated from the initial data provided

```
%% Neural Network
TrainData = imageDatastore('./train','IncludeSubfolders',true,'
    LabelSource', 'foldernames');
uLabelsTrain = unique(TrainData.Labels);
nClsTrain = numel(uLabelsTrain);
TestData = imageDatastore('./test','IncludeSubfolders',true,'
    LabelSource', 'foldernames');
uLabelsTest = unique(TestData.Labels);
nClsTest = numel(uLabelsTest);

fSize = 3;
nFilters = 32;

layers = [imageInputLayer([32,32,3],'Name','Input layer');
    convolution2dLayer(3,32,'Name','conv layer');
     reluLayer('Name','relu layer 1'); maxPooling2dLayer(2,'Stride'
         ,2,'Name','2x2 MaxPooling');
     fullyConnectedLayer(512,'Name','fullyconnected layer 1');
     fullyConnectedLayer(nClsTrain,'Name','fullyconnected layer 2');
     softmaxLayer('Name','softmax layer'); classificationLayer('Name
         ','class layer')];

options = trainingOptions('sgdm','InitialLearnRate', 0.0002,...
     'MaxEpochs',10);
net = trainNetwork(TrainData, layers, options);

labelTest = classify(net, TestData);
Accuracy = mean(labelTest == TestData.Labels);
disp(['Classification accuracy (Test) = ' num2str(100*Accuracy,3) '
    %'])

lgraph = layerGraph(layers);
```

```
figure
plot(lgraph);


disp(['Risk empirical functional = ' num2str(1-Accuracy)])
m = length(labelTest);
deltas = [0.1,0.05,0.01,0.001];
for i =1:4
    delta=deltas(i);
    epsilon = sqrt(log(2/delta)/(2*m));
    disp(['With delta = ' num2str(delta) ', we obtain epsilon = '
        num2str(epsilon)])
%     disp([ num2str(delta) ' & ' num2str(epsilon) '\\']) % Para
    tabla en
%     latex
end
```

6.2 Neural network for spectrogram generated from the new data

```
%% Neural Network
TrainData = imageDatastore('./train3','IncludeSubfolders',true,'
    LabelSource', 'foldernames');
uLabelsTrain = unique(TrainData.Labels);
nClsTrain = numel(uLabelsTrain);
TestData = imageDatastore('./test2','IncludeSubfolders',true,'
    LabelSource', 'foldernames');
uLabelsTest = unique(TestData.Labels);
nClsTest = numel(uLabelsTest);

fSize = 4;
nFilters = 128;
layers = [imageInputLayer([32,32,3],'Name','32x32x3 Input layer')
    ...
            convolution2dLayer(fSize,nFilters,'Name','128 4x4 (stride
                1) conv layer')...
            reluLayer('Name','relu layer 1')...
            maxPooling2dLayer(4,'Stride',4,'Name','4x4 (stride 4)
                maxpooling layer')...  % 8x8x128
            convolution2dLayer(fSize/2,nFilters*2,'Name','256 2x2
                conv layer')...
            reluLayer('Name','relu layer 2')...
            maxPooling2dLayer(2,'Stride',2,'Name','2x2 (stride 2)
                maxpooling layer')... % 4x4x256
            fullyConnectedLayer(4096,'Name','4096 fullyconnected
                layer')...
            reluLayer('Name','relu layer 3'),...
            fullyConnectedLayer(1024,'Name','1024 fullyconnected
                layer')...
```

```
21            reluLayer('Name','relu layer 4')...
22            fullyConnectedLayer(nClsTrain,'Name','3 fullyconnected
                 layer')...
23            softmaxLayer('Name','softmax layer')...
24            classificationLayer('Name','class layer')];
25
26 options = trainingOptions('sgdm','InitialLearnRate', 0.0002,...
27     'MaxEpochs',20);
28 net = trainNetwork(TrainData, layers, options);
29
30 labelTest = classify(net, TestData);
31 Accuracy = mean(labelTest == TestData.Labels);
32 disp(['Classification accuracy (Test) = ' num2str(100*Accuracy,3) '
      %'])
33
34 % lgraph = layerGraph(layers);
35 % figure
36 % plot(lgraph);
37 %lgraph = connectLayers(lgraph,'relu_1','add/in2');
38
39 disp(['Risk empirical functional = ' num2str(1-Accuracy)])
40 m = length(labelTest);
41 deltas = [0.1,0.05,0.01,0.001];
42
43 for i =1:4
44     delta=deltas(i);
45     epsilon = sqrt(log(2/delta)/(2*m));
46     disp(['With delta = ' num2str(delta) ', we obtain epsilon = '
          num2str(epsilon)])
47 %     disp([ num2str(delta) ' & ' num2str(epsilon) '\\']) % Para
      tabla en
48 %     latex
49 end
50
51 m = floor(0.1*length(labelTest));
52 rng(800)
53 L = randperm(length(labelTest),m);
54 Loss = zeros(m,1);
55 for i = 1:m
56     Loss(i,1) = labelTest(L(i))~=TestData.Labels(L(i));
57 end
58 Loss = mean(Loss);
59 disp(['Risk empirical functional = ' num2str(Loss) '%'])
60 deltas = [0.1,0.05,0.01,0.001];
61 for i =1:4
62     delta=deltas(i);
63     epsilon = sqrt(log(2/delta)/(2*m));
64     disp(['With delta = ' num2str(delta) ', we obtain epsilon = '
          num2str(epsilon)])
65 %     disp([ num2str(delta) ' & ' num2str(epsilon) '\\']) % Para
```

```
    tabla en
% latex
end

confusionmat(labelTest,TestData.Labels)
```

Bibliography

[1] R. O. Duda, G. Stork, P. E. Hart. *Pattern classification and scene analysis*, Wiley, 2000.

[2] V. Vapnik. *The Nature of Statistical Learning Theory*, Springer, 2000.

[3] O. Bousquet, S. Boucheron, G Lugosi. *Introduction to statistical learning theory*. In Summer School on Machine Learning (pp. 169-207). Springer, Berlin, Heidelberg, 2003.

[4] I.Y. Tyukin, A.N. Gorban, A.A. McEwan, S. Meshkinfamfard, L. Tang. *Blessing of dimensionality at the edge and geometry of few-shot learning*. Information Sciences, 564, pp.124-143, 2021

[5] I.Y. Tyukin, A.N. Gorban, M.H. Alkhudaydi,Q. Zhou. *Demystification of Few-shot and One-shot Learning*, 2021. arXiv preprint arXiv:2104.12174. (to appear in IEEE IJCNN 2021)