Netherlands Forensic Institute
Ministry of Security and Justice

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Exploring deep learning to improve allelic peak calling in forensic DNA analysis

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

MASTER OF SCIENCE
in
APPLIED MATHEMATICS

by

Myrte van Belkom

Delft, the Netherlands
August 2021

Netherlands Forensic Institute
Ministry of Security and Justice

MSc THESIS APPLIED MATHEMATICS

"Exploring deep learning to improve allelic peak calling in forensic DNA analysis"

Myrte van Belkom

Delft University of Technology

Responsible professor

Dr. ir. L.J.J. van Iersel
Associate professor of Optimisation

Daily Supervisors

Dr. ir. L.J.J. van Iersel
Associate professor of Optimisation

Dr. ir. R.J.F. Ypma
Principal scientist at Netherlands Forensic Institute

Other thesis committee members

Dr. J.P. Gonçalves
Assistant professor of Bioinformatics lab

August, 2021

Delft

Abstract

When processing a trace DNA sample at the Netherlands Forensic Institute, an STR electropherogram can be created. An analyst uses this electropherogram and analysis software to read out peaks signifying DNA. After analysis, the DNA profile is used in the interpretation process, which can include the comparison to a reference DNA profile of a person of interest. The software that is currently being used for profile analysis is threshold-based and the process includes the intervention of trained analysts. To further automate (allelic) peak identification in STR electropherograms, as well as to increase efficiency and uniformity, neural networks were studied and applied. Previous work by Duncan Taylor and David Powers provided a proof of concept using a simple fully connected neural net. After reviewing literature, the U-net was selected to be used in this thesis. Training U-net on electropherograms proved successful and achieved a $\sim 95\%$ accuracy on the per-pixel labels. However, translating the per-pixel output to alleles was more difficult than expected, so an upper bound on the score was calculated. The upper bound got close to an analyst's performance and demonstrated the potential of this method.

Preface

This thesis was written at the Netherlands Forensic Institute as a graduation thesis from TU Delft. Its goal is to explore deep learning methods to apply on forensic DNA analysis. Hence, the topic of this thesis is multidisciplinary: both the fields of biology (division Biological Traces at the NFI) and computational modelling (team Forensic Big Data Analysis at the NFI) are visited. The thesis committee members are Leo van Iersel (supervisor at TU Delft), Rolf Ypma (supervisor at NFI) and Joana Gonçalves (Delft Bioinformatics Lab).

This specific application of neural networks to STR electropherograms is quite new, so the number of peer reviewed articles regarding comparable research was limited. Based on the few articles available, and following advice from experts in deep learning at the NFI, a U-net was selected. To implement this, the DNA composition (alleles) had to be translated to per-pixel labels on the full electropherograms (which can be seen as a set of 6 graphs). The U-net was trained and achieved $\sim 95\%$ accuracy on the per-pixel labels. Translating this back to DNA composition (alleles) proved challenging, because of the raw type of electropherogram data used. While future research may focus on improving this translation, we decided to calculate an upper bound of how well this method could possibly perform if the translation were optimised. The upper bound on the score performed similarly to that of a trained analyst. This shows the potential of U-nets, and deep learning in general, when analysing electropherograms for forensic DNA analysis.

The NFI was an amazing host institute. During my time there, they organised a multitude of meetings and presentations where their scientists gave talks about their research. As someone who is interested in all fields of forensic science, this was a great opportunity for me to learn about the NFI and forensic work in general. The people I was working with, from team FBDA and division BiS, were very helpful and welcoming. They were always open to questions, and made me feel like a part of the team. During the second lockdown, I had a hard time. It was mainly because of the help I received at the NFI that I was able to continue my thesis. For this, I would like to thank Corina and Francisca, and especially my supervisor Rolf. Without them, I would not have been able to finish my thesis during this pandemic.

Myrte van Belkom
August 10, 2021

Contents

Introduction

This thesis was carried out at the NFI: the Netherlands Forensics Institute. The NFI is a government institute whose main goal is to provide forensic analysis for the police and public prosecutor's office. All work at the NFI is divided into three core activities: case work (main goal), innovation (Research&Development) and sharing knowledge (e.g. seminars, internships). These categories can even overlap; the subject of this thesis falls into the R&D category, but if it is successful, it may be used in actual case work.

Problem statement

Every day hundreds of DNA samples are processed at the NFI. This task requires a trained DNA analyst to read out the peaks corresponding to actual DNA from a measurement (STR electropherogram), which can be time-consuming and readings can vary per analyst. A trained analyst views the electropherogram in software (for instance Genemarker™ , GeneMapper™ or OSIRIS™) which filters out some artefacts using a set of rules. For example, it uses information on how high an artefactual peak (stutter) is expected to be, relative to the allelic peak causing it. This makes Genemarker™ able to filter out the most common artefacts before the analyst looks at the electropherogram.

At the NFI, a simple automated method has previously been developed to do part of the analysis, and filter out some more common artefacts. The current automated algorithm (*Snelle ID-lijn*) is built upon the existing software; Genemarker HID auto v2.9.8™ . It would be preferable to develop an automated method on data not dependent on the type of software used. Other institutes may use different software, and the NFI may decide to use different software in the future too. This means that the data to be analysed is quite raw and has no added information.

In 2016, it was shown that neural networks, a type of deep learning, could be trained to analyse an electropherogram [1], [2], [3]. This was not yet a perfect solution, and more of a proof of concept. However, it is very promising that the raw data could possibly be analysed decently without a DNA analyst or advanced, threshold-based preprocessing from a software package like Genemarker™ . This would increase uniformity of the results and efficiency of the analysis. The goal of this thesis is to explore whether neural networks could enhance forensic DNA analysis at the NFI.

Overview

In the next chapter, Chapter 1, the necessary background knowledge in DNA analysis will be introduced. Then, in Chapter 2, we will look into the available data and explain in detail how the DNA data files are generated and how they are being used. Chapter 3 contains both a brief introduction into deep learning, specifically neural networks, as well as a literature study on why the U-net was selected. In Chapter 4 we discuss the settings of the U-net and format of the input. Chapter 5 contains the methods used to obtain the results in Chapter 6. Chapter 7 and 8 respectively, cover the conclusion and discussion.

1 Background: DNA

We are looking into a method to enhance forensic DNA analysis. The goal of forensic DNA analysis is to identify the source (a person) of a DNA trace. The method we are applying to this biology subject, is deep learning; a topic in computer science. That makes the topic of this thesis in the field of bio-informatics. To understand this topic, both knowledge in the field of biology (DNA) as well as the field of computer science (deep learning) is needed. We will very briefly introduce the necessary background information on the DNA side in this chapter. The 3rd chapter will be dedicated to explaining the deep learning side of things.

In this chapter, first the basics of DNA are visited. Then we quickly move on to forensic DNA extraction and analysis. Finally, I will explain the difficulty with reading out the data. Not all of the information is relevant to the project work itself, but it is needed for interpretation of the results.

1.1 DNA

DNA can be found in nearly every cell of your body (pages 139-140 in [4]). Each cell nucleus (besides a gamete's nucleus) has the same 23 chromosome pairs. Per pair, one chromosome is inherited from each parent. All information stored in these chromosomes makes you practically unique, and hence it can be used to identify a person.

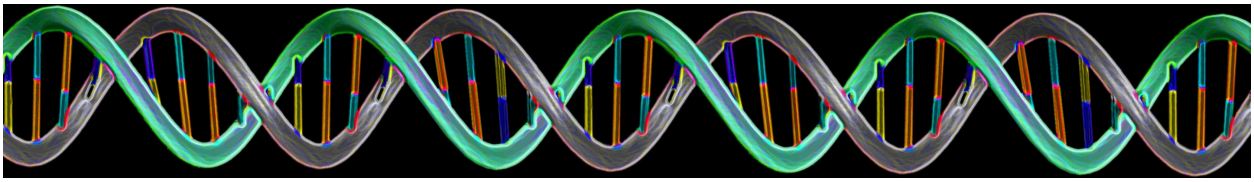


Figure 1: DNA[1]

DNA is stored in the chromosomes, and is built from two strings that wrap around eachother in the shape of a (double) helix (see Figure 1). These two sides are connected to eachother by nucleotides, represented by the letters A(denine), C(ytosine), G(uanine) and T(hymine). The two sides have opposite nucleotides. The A is always opposite to the T, and the C always opposite to the G. One pair of two opposing nucleotides is called a base pair.

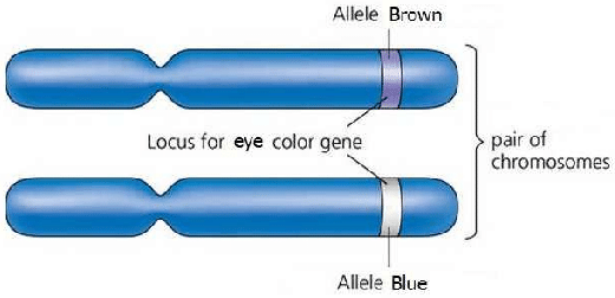


Figure 2: Visualisation of a pair of chromosomes and one set of alleles[2]

On chromosomes, there are certain fixed positions, loci (page 140-143 in [4]), where the DNA can differ in known ways (see Figure 2). On some loci the different varieties of allele differ from each other by the number

[1] nisenet.org
[2] researchgate.net

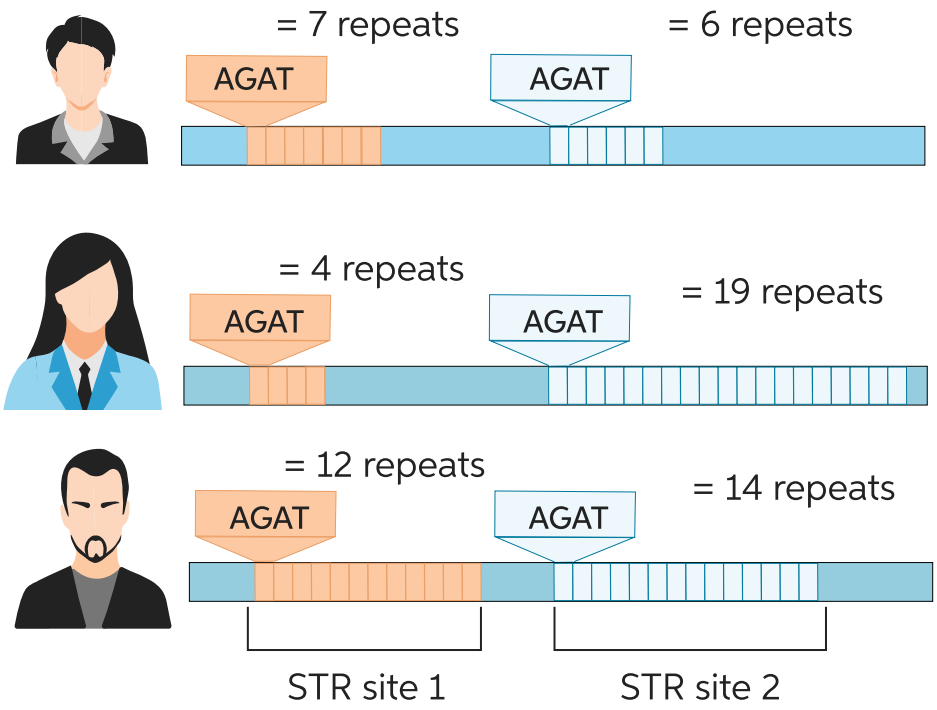


Figure 3: Visualisation of how alleles show variation between persons [3]

of repetitions of a short sequences of nucleotides: Short Tandem Repeats (STR). The number of repeats are called alleles. Since these STRs are easily identified and show variation over the population, they are ideal to use for DNA analysis. For an example, see Figure 3. Here, identifying only one allele would be enough to discern between the three people. The more alleles you can identify, the more certain and unique your identification is. But how would you obtain this information from a DNA sample?

1.2 Lab techniques

When trying to identify a person of interest from a human biological trace, forensic scientists follow a couple of steps (pages 142-144 in [4]). First, the DNA is extracted as purely as possible and quantified. Then PCR (Polymerase Chain Reaction) is carried out to label and copy only the STR regions of interest (Section 1.2.1). Finally CE (Capillary Electrophoresis) is performed to separate the amplified fragments based on label (dye color) and length (basepairs), which will be discussed in Section 1.2.2. A graph of this feedback is called an electropherogram, as we will see in Section 1.2.3.

Note: The NFI uses a 6-dye kit and software called Genemarker™ . Other companies may use a 5-dye kit, or a different software like Genemapper™ or OSIRIS™ .

1.2.1 PCR

During the PCR, five different colour fluorophores are added to specific STRs to enable distinguishing between fragments in the same range of fragment length. A fluorophore is a chemical compound which emits light after excitation (fluorescence). A sixth fluorophore is added to the size standard, a term which I will explain later on in this chapter, during the next process: electrophoresis. See also the diagram in Figure 4

[3] chegg.com

for more details of the PCR.

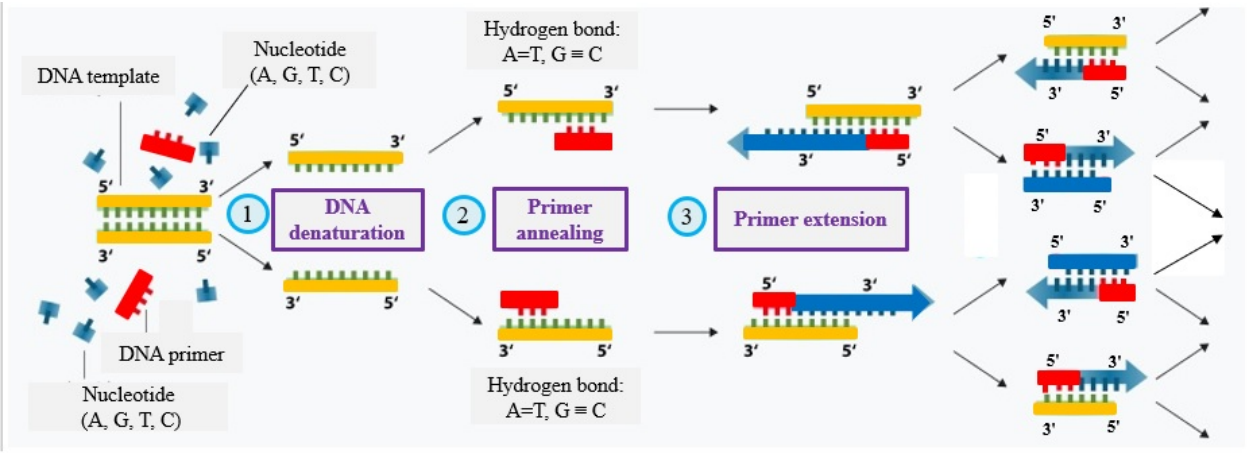


Figure 4: PCR[4]

The goal of PCR is to multiply certain parts of DNA while leaving the rest. First, the double DNA strings are pulled apart to access the nucleotides (DNA denaturation). *Primers* are small bits of DNA which link to a region of interest and act as a starting block for the replication (primer annealing). The mixture is also filled with loose nucleotides which rebuild the other half of the DNA string from the primer on (primer extension). When the region of interest has been copied, the (now again) double strings are pulled apart and the process repeats on both original and copied parts of DNA.

1.2.2 CE

Finally CE (Capillary Electrophoresis) is performed on the resulting mixture of PCR to separate the amplified fragments based on label (dye colour) and fragment length (basepairs). The sixth dye is added to the *size standard*, which will be further explained in Section 1.2.3. The sample is led through a very thin tube and hit with a laser. The fluorescent feedback is read by a sensor, resulting in 6 different graphs; one for each dye. This fluorescent feedback is locally very high for the copied STRs and shows up as a peak in the data. So, at this point we have 6 measurements of relative fluorescent units (rfu) against time.

1.2.3 Electropherogram

We now have this graph that signifies which alleles are present, but we cannot see a lot from the raw data, since its horizontal axis is time. The next step is to rescale the measured data's horizontal axis (time) into fragment length, so that the alleles can be identified. The longer a string of DNA is (the more nucleotides it has), the slower it passes through the tube, and the longer it takes before the DNA passes by the laser. Using a so-called *size standard*, a known sample with its own fluorescent dye, which is run through the machine at the same time, this rescaling is possible. The peaks in the size standard are very easily distinguishable from the baseline and have fragments of known sizes. So the time at which these peaks show up, can be linked to a certain length in fragment size. This relation between the time and fragment length can then be applied to all other dyes as well.

To link all peaks to alleles in the unknown sample, a *ladder* is used. A ladder is a mixed sample which (theoretically) contains a mixture of all possibly occurring alleles for each locus. All steps in this and the last paragraph are usually automatically performed by Genemarker™ : a genetic analysis software used at the NFI. The data can be visualised in a graph with the 6 dyes on the vertical axis: the STR electropherogram. An STR electropherogram shows, for each dye, fluorescent feedback in rfu (on the vertical axis) against

[4] case.ntu.edu.tw

fragment length in base pairs (on the horizontal axis). An example of how a complete electropherogram looks, can be found in Figure 5.

1.3 Profile analysis

When the electropherogram has been loaded into Genemarker™ (with all preprocessing applied), a forensic analyst identifies which peaks he deems to be allelic (caused by DNA that was present in the sample), and which are artefactual. Artefacts are caused by small errors in the DNA-replication process, and lead to peaks in the electropherogram not related to the actual DNA. The two most common types of artefact are: stutter and pull-up.

A stutter occurs when the DNA string folds a little and the copied string skips one STR (Figure 6). A stutter can occur on both sides of the DNA string, so it can cause a piece of DNA to be one STR shorter or longer than it was supposed to be. This can even happen with two STRs difference, but this is less common than with 1 STR. More than 2 STR difference was regarded too rare to even consider in this study. If this mistake occurs often during PCR, it shows up in the electropherogram as a small peak right before or after an allelic peak: a stutter. Many examples of stutter can be found in Figure 5, almost each allelic peak in the green dye has a stutter in front of it. This is called a backward stutter, and is the most common type of stutter. A stutter artefact can actually be used to confirm that the *following* peak is not artefactual, but allelic.

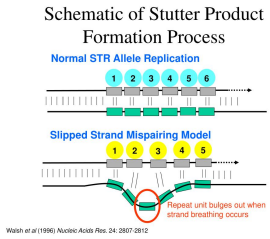


Figure 6: Visualisation of how stutter can occur [5]

Pull-up occurs because the 6 colours' spectra overlap partly. Some high peaks can appear as smaller peaks in other dyes, even though they only correspond to actual DNA in one dye. To find this type of artefact, you need to look vertically throughout the different dyes. You can see an example in Figure 5 in the sixth orange dye. This is the size standard, which should only have peaks at set intervals. The small peaks at 168.30, 338.10 and 420.10 basepairs are all results of pull-up, most likely from the fifth purple dye.

Genemarker™ filters many of the errors out of the graphs according to static and dynamic thresholds, before an analyst even looks at the data. There is a lot of noise at the almost zero level of rfu, which is not visible in Figure 5 because of this preliminary filtering and the scale of the y-axis. Genemarker™ also automatically calls the allele of each peak, or whether a peak is present but does not seem to correspond to an allele (OB: Out of Bin). An analyst can then decide what to do with such a peak. On the other hand, Genemarker™ also checks for the most common types of artefact, and specifically does *not* label these artefactual peaks: stutter filters. Genemarker™ has information on which stutters could occur per allele, and how high (relative to the allelic peak) the stutter peak is expected to be. All this specific information enables Genemarker™ to recognise most stutter peaks accurately, so the analyst does not have to.

After all this preliminary analysis, an analyst decides which alleles to include or remove. The analyst then reports only the alleles present and heights of each peak corresponding to that allele.

[5] slideserve.com

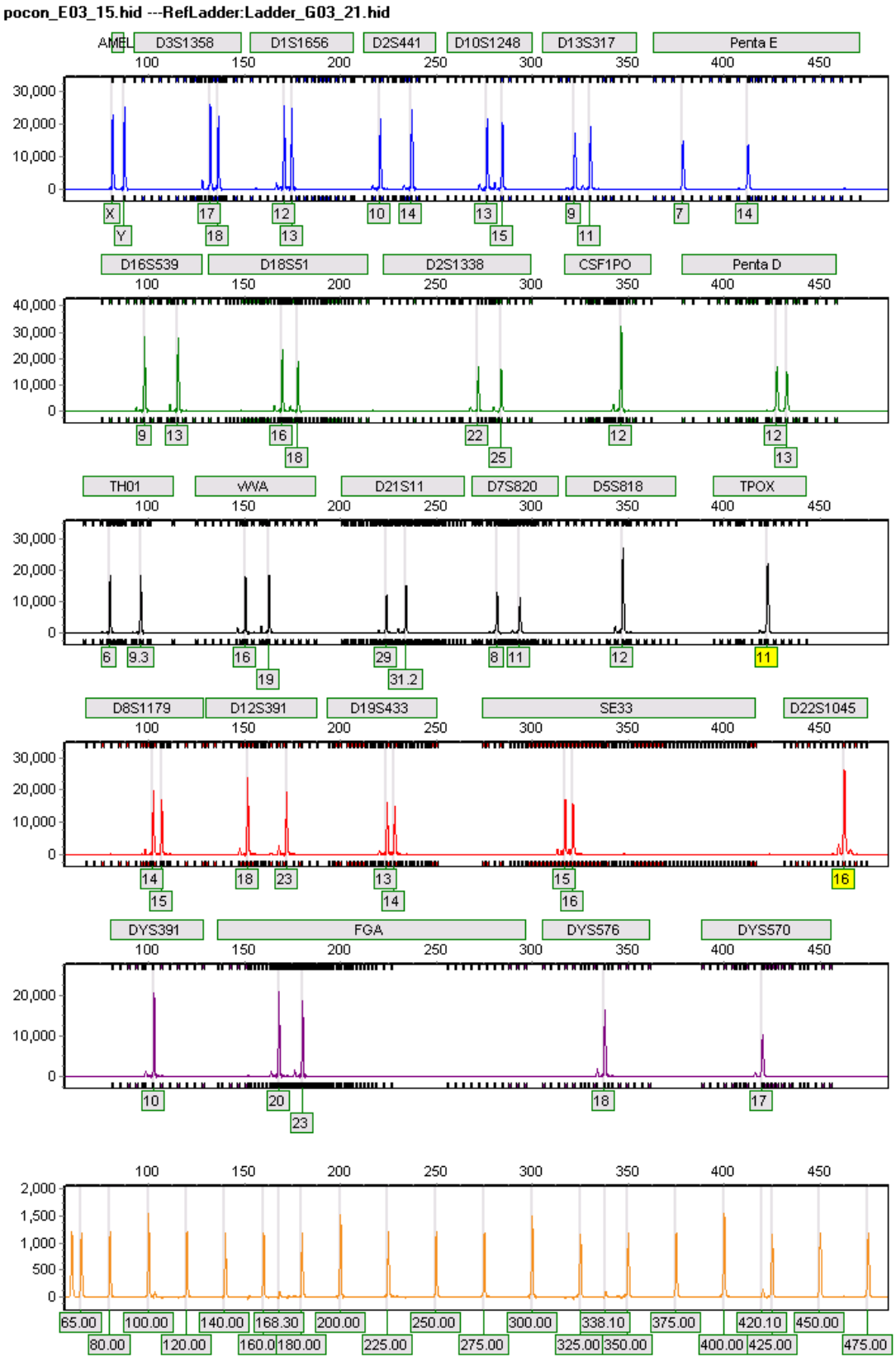


Figure 5: Example of an electropherogram in Genemarker ™

1.4 Interpretation

Based on the information the analyst has reported, the reporting officer interprets these results (pages 145-149 in [4]). They make an estimation of the number of contributors. If there are multiple donors, and if possible, they also deconvolve the mixture. Deconvolution is the splitting of a mixture profile into the profile per donor, usually at least the main donor's profile is deduced. Then they decide whether a profile is suitable for comparison against other reference profiles within the case, or for inclusion in a DNA database. If the profile is not regarded suitable, additional analyses may be performed on the sample, e.g. analysing a replica of the sample, decreasing thresholds or using a different kit.

If the profile is good enough, the reporting officer performs weight of evidence calculations. These are reported as a likelihood ratio. The probability of observing the evidence given that hypothesis H_1 is true, divided by the probability of observing the evidence given that hypothesis H_2 is true. Generally, H_1 is the prosecution hypothesis, e.g. the person of interest and an unknown person contributed to the trace. And H_2 is generally the defense hypothesis, e.g. two unknown contributors, unrelated to the person of interest, contributed to the trace.

1.5 Back to the project

This is how forensic DNA analysis is currently being carried out. A problem in this process, is that all analysis has to be manually checked. This is very costly, time-consuming and can yield differences between different analysts. FBDA (the Forensic Big Data Analysis team at the NFI) developed a simple automated method built upon Genemarker HID auto™ to automate some extra steps of filtering out artefacts and calling alleles. However, further automation and omitting a threshold-based analysis software may further optimise the DNA profiling process.

Neural networks have been shown to be effective in many problems, and have also been successfully applied to electropherograms [1], [2], [3]. So with the aim to streamline the process of forensic DNA analysis, neural networks were examined for use on electropherogram data at the NFI. The exact extent of previous research will be discussed more thoroughly in Chapter 3. But first, the data files available at the NFI will be introduced in Chapter 2.

2 Data

This chapter focuses on the data files available and how they are being used. Almost everything covered in this chapter was carried out during a separate internship, and not part of the thesis work. However, a summarised version of the internship report will be included, since the knowledge obtained is very useful for the rest of the thesis. The code developed during this time will also be used in my thesis. All coding was done in Python 3.8[6] using the PyCharm IDE[7]. A full copy of my internship report is included in Appendix B and all code developed during this thesis can be found on a GitHub repository[5].

We begin by describing the goal of the internship project work in Section 2.1. Section 2.2 will take you through the information inside the data files and how it is being used. Section 2.3 covers some even more in-depth use and application of the files, and the beginning of the project work of my thesis. It will be clearly stated from which point on the work will be part of the thesis.

2.1 Goal

Before we look at the content of the data files, it is important to know which information we need. Previous research by Duncan Taylor and David Powers [1] on training a neural net, used a label per measured data point. These electropherograms were annotated by hand and considered five different labels for each point: baseline, allele, (backward) stutter, forward stutter, and pull-up. In Figure 7, a single labeled dye of two different electropherograms is shown.

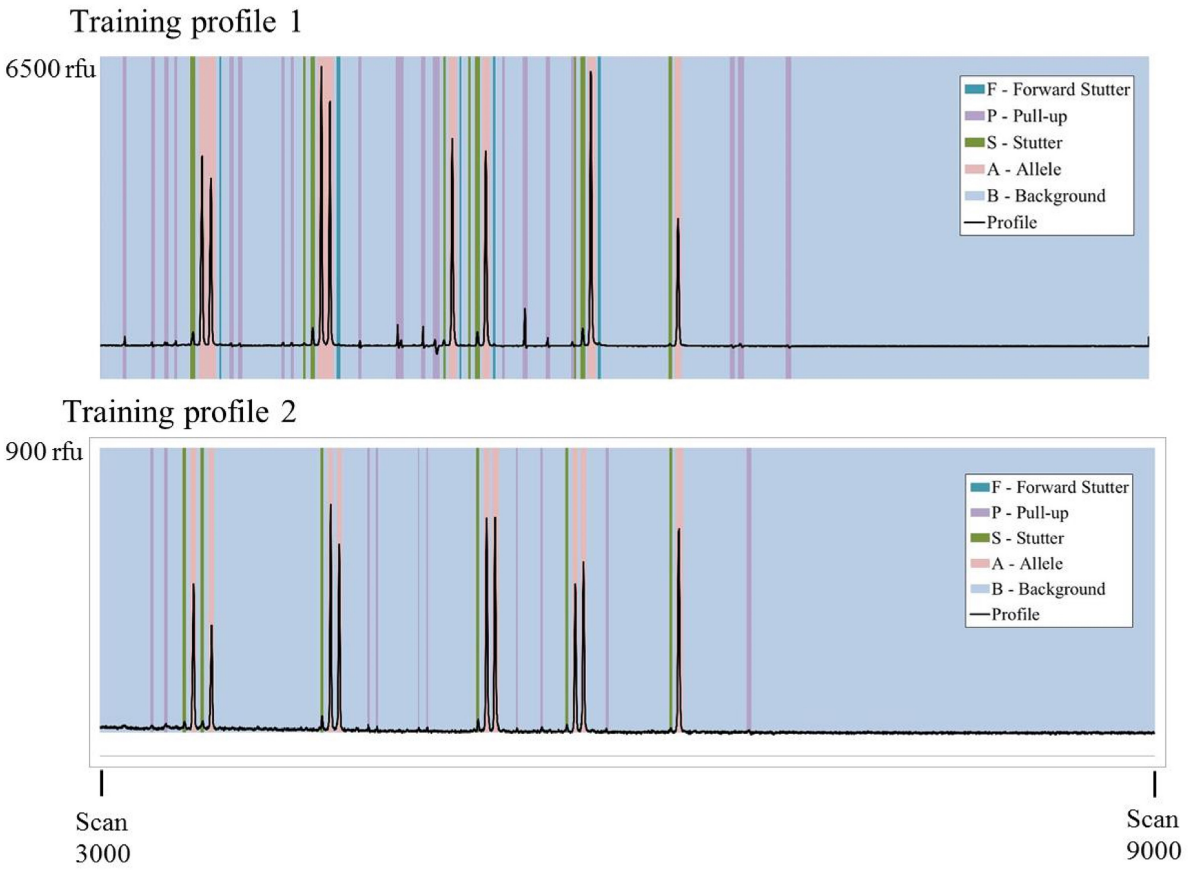


Figure 7: Labels used by Duncan Taylor and David Powers in [1]

[6] python.org
[7] jetbrains.com/pycharm

So the first goal is to obtain similar labels and similarly shaped data: we want to be able to label each point of measurement in the electropherogram.

2.2 Data files

Duncan Taylor and David Powers had access to annotated electropherograms per pixel for 5 different labels. First, we will introduce the available data files for this thesis to see how close to this format we can get. At the NFI an R&D dataset of electropherograms was available to use. This means the electropherograms were not created from case data, but from an experimental set of DNA samples. Each sample was replicated and run 3 times, and for each of these measurements, we have the following data.

- .hid Raw data from genetic analyser
- .txt Trace data of DNA samples (raw and sized)
- .csv Analysts' identified alleles and peak heights

And the following files with information needed for creating the labels:

- .csv Donors' DNA profiles
- Microsoft Word™ document describing the composition of the DNA mixtures as published in [6]
- .xml Genemarker™ panel info

It would be ideal to be able read out the .hid files, so there would be no dependency on Genemarker™ at all. However, this data is extremely raw measurement data; the files are are not understandable to the human eye, they contain lots of random whitespace and characters. So we use Genemarker™ to read the .hid file and then immediately export the data without applying any advanced filters or analyses (besides the resizing of the axes to fragment length). There are two export options that do not require any advanced analysis from Genemarker™ : raw trace data and sized trace data.

The last file listed, is the Genemarker™ panel .xml file. This file is used for a number of things. It contains a huge amount of information, on the fragment lengths of alleles, which alleles are on which locus, which locus is measured with which fluorescent dye, and much more. It was used for plotting purposes, as well as to relate allele names to fragment size (position on the horizontal axis).

2.2.1 Raw versus sized trace data

The first question of my internship was: what is the difference between the two export options in Genemarker™ : raw trace data and sized trace data. To discover the difference, both were plotted in the same figure for each dye in Figure 8.

The first difference is that the length of the sized data is much shorter than the raw data for each sample (about 6000 measured data points compared to 9000). This has to do with both a resizing and a translation. The precise number of measured data points differs between files within both sets as well. The sized data measurement only starts after the *primer dimer*, where the raw data measurement starts a lot earlier. *Primers* are small single-stranded DNA that act as starting points for PCR. *Primer dimer* is the fluorescent feedback of all the primers that were in the DNA sample for replication during PCR, but were not used in the process. These primers are all very short and light structures, and easily distinguishable from the actual peaks. Because of their small fragment length, they show up as the very first peak, since they reach the laser first. Each measurement has such a primer dimer blob as its first peak.

After removing the same first part of measurement from the raw data, there are still differences between the two data types. The horizontal axis had been rescaled for the sized data, and not for the raw data. The size standard always shows peaks at the same locations, as can be seen in Figure 5 as well. Using the size standard, we discovered that the sized data has precisely 10 data points per base pair. So at the 100th point, a peak would signify a fragment length of 10 basepairs. Because this rescaling is a nonlinear rescaling of the raw trace data, we decided to use the sized trace data from here on out.

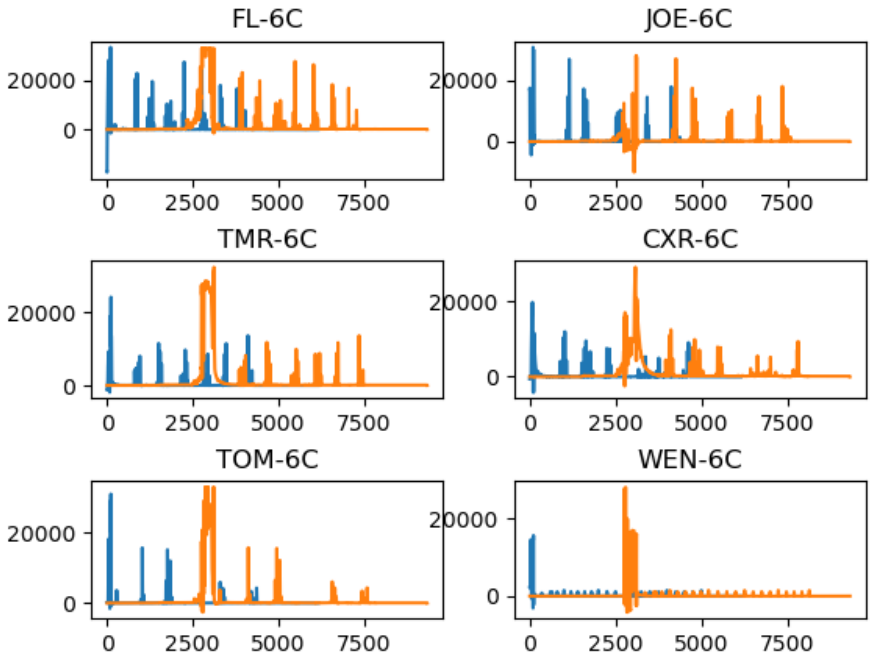


Figure 8: All six dyes of the raw (orange) and sized (blue) trace data.

2.2.2 Profile composition

Now that it was possible to interpret this, still quite raw, sized data, the labels could be added. There are two files that contain annotated alleles of the electropherogram. The first option is to use the analysts' identified alleles, which were ordered in simple .csv files per sample. The analyst reports the alleles and their heights (as seen in Genemarker™). There may be small errors in the reading, but it is very straight-forward data to use.

Second option is to calculate the theoretical expected alleles from the known sample composition. These are *expected*, because it is possible that an allele does not show up in the electropherogram. There are .csv files of each donor's DNA profile, as well as a text document with explanation on the composition of each mixture. All of the experimental data was composed from known donors in known proportions. These samples are coded by a short sequence denoting the amount of different donors, the ratios of DNA between donors and which set of donors the sample came from.

In both cases, there is a little extra information available than strictly necessary. The only thing that will be used from these files is the list of alleles present in the mixture. These should be visible in the electropherogram as peaks. Using the Genemarker™ panel file, the alleles can be related to the horizontal axis (fragment length). The specifics of this are discussed in the next section: Section 2.3

2.3 Adding labels per data point

The previous sections of this chapter were research carried out during the internship. The next problem is the first problem studied as part of the thesis: now that the alleles present in a mixture could be obtained, we needed a way to find the output peak of that allele in the electropherogram.

The data we obtained are the alleles in a mixture, either theoretical expected or identified by analyst. Then the Genemarker™ panel file gives an approximate location of where the peak should be visible (a *bin*) on

the horizontal axis. This gives us at least a general idea of where the peak is, but the full peaks are wider than the bins themselves.

As described in Section 2.1, each measured point is supposed to get its own label, so some kind of peak selection algorithm is needed to identify all the single points constituting each allelic peak. Furthermore, Duncan Taylor and David Powers used five different labels in their papers, but with the available files only two labels can be added: allelic peak or not an allelic peak.

2.3.1 Peak selection algorithm

An example of the output of the algorithm on a full electropherogram can be found in Figure A.1 in Appendix A. We only highlight some details of this figure within this chapter in Figures 9 and 10. The vertical blue lines show the bins given in the Genemarker™ panel file, and the green area shows the entire peaks selected. To explain the way this was designed, we look at a few details of this image.

The allele bins given in the panel file, are not completely accurate for the location of the resulting peak (see Figure 9). Some bins are shifted too much to one side, and only contain part of the peak. This means it cannot be assumed that the top of the peak is within the bin. Upon inspecting the data, we noticed that each bin does intersect with at least a part of the corresponding peak.

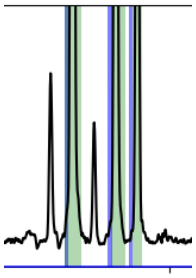
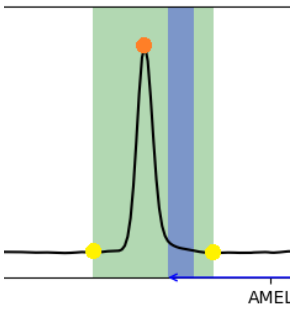
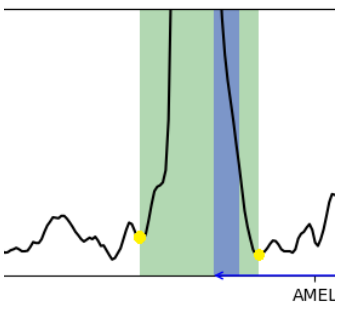


Figure 9: Detail of Figure A.1

Based on this fact, a simple algorithm was designed. We use the knowledge of which alleles should be visible, and get the allele bin from the Genemarker™ panel file. Then, the highest point of measurement within the allele bin is found. We check whether we can move in either direction to increase this value (possibly moving outside of the bin). We keep on updating this value until it has found the local maximum, so until the top of the peak has been found.



(a) Zoomed out to highlight top of peak, rfu range is -2000 to 20000 rfu



(b) Zoomed in to highlight start and end of peak at baseline, rfu range is -50 to 500 rfu

Figure 10: Visualisation of peak selection algorithm. Green background: region selected by algorithm as peak. Blue background: allele bin. Orange dot: top of peak. Yellow dots: resulting start and end points of selected peak.

We make use of another fact, which is that the measurement data is extremely smooth. From the top of the peak found, we select the left and right end points as long as the values are monotonously decreasing. To illustrate how well such a simple algorithm works: for all images in this subsection, the peaks were selected by this algorithm. It only requires the information from the Genemarker™ panel file, and performs quite well. In Figure A.1 in Appendix A we see a full electropherogram where all peaks have been annotated.

We decided to add this label for peaks in the size standard (6th dye) as well, since we want to train our network to detect peaks. It might be very confusing to the network if we do feed it the size standard with very clear peaks, and tell it not to detect those peaks. This would just make training unnecessarily more difficult. We do want to include the size standard in our input, because it may show pull-up from other dyes, or cause pull up in other dyes.

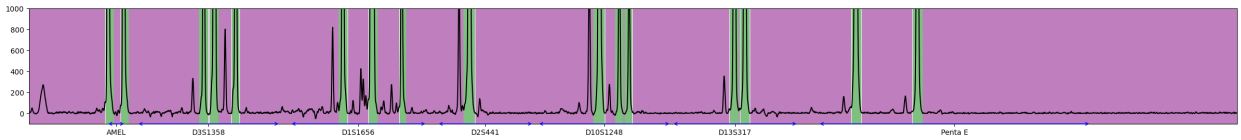


Figure 11: Example labels for all points in top dye of electropherogram. Green: peak, purple: background

Then, because no other labels are available, every other point is set to the *background* label. An example of the result of this algorithm on a single dye, can be seen in Figure 11. This means we are now able to add a per-pixel label to the entire electropherogram, as discussed in Section 2.1. This gives similarly shaped data to what David Powers and Duncan Taylor used, but simpler because we use less labels. It is actually a lot easier to use, since two different labels means it is binary data (peak/background). Both the measurement and the labels can be represented as a $n \times 6$ matrix, where the label matrix is a boolean array.

3 Neural Networks

Now that we have seen the available data, and have sufficient background knowledge in DNA, it is time to look at the computational modelling side: neural networks. We will begin by explaining the basics of deep learning and what makes up a neural network in Section 3.1. Then we will review other applications of neural networks in the field of bio-informatics in Sections 3.2 up to 3.3. From this, the choice for a U-net will be explained in Section 3.3.

3.1 Background: Deep learning

Deep learning[7] is a class of machine learning based on artificial neural networks. Machine learning is the process of letting a computer come up with its own algorithm to do a certain task by feeding it a lot of data. Deep learning generally requires less intervention from the user and less structured data. The user does not have to select features, the network trains to both extract features and subsequently classify the data.

If the data is labeled, that means the desired output is specified for training. This is called supervised learning. If the algorithm is not given labels or desired output, it is called unsupervised learning. As an example, think of labeling movies by genre. If we gave the algorithm a set of genres to pick from (horror, comedy, romance) and then optimised its decision making, that would be supervised learning. If we just gave the algorithm a set of movies, without specifying the genres and we ask it to group them together, it would be unsupervised. In unsupervised learning, we do not tell the algorithm which labels or groups should be selected. This means it could output genres no one has ever heard of. It is a very powerful tool to get to know your data. If you do have more information about your data, it is often better to apply supervised learning. In our case as well, since we have a lot of information on our data. We already know we are interested in the alleles present in a mixture, the algorithm does not need to find that out on its own. In this thesis we only consider supervised learning, and more specifically neural networks.

3.1.1 Neural network

A (artificial) neural network is modeled after the brain (hence the *neural*). It consists of a set of neurons or nodes, which are connected by edges with weights. The nodes are subdivided into layers. The first one (green in Figure 12) is the *input layer* which has one node per point of input. An image of 1024 pixels for example, would require 1024 input nodes (3096 if the image is RGB). The input layer is followed by at least one, but often more *hidden layers* (blue in Figure 12). The final layer is the *output layer* (yellow in Figure 12). This often gives a probability to the input belonging to a certain class. As we will see later on in Section 4, our network also ends in a single node. The probability at this node corresponds to the probability of there being an allelic peak at that location.

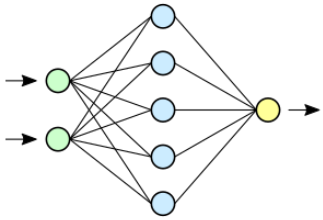


Figure 12: Illustration of a neural network[8]

The output layer can also be more than one node, for example in a multiclass labeling problem like [1]. They distinguish 5 different classes for each pixel of the electropherogram (allele, background, backward stutter, forward stutter, pull-up). This will be more extensively covered in Section 3.2. To give a simpler example of a multi-labeling problem, see Figure 13.

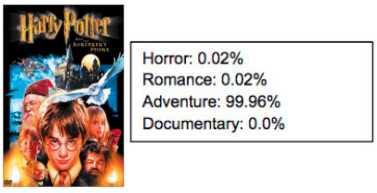


Figure 13: Example output of multi label classification problem[9]

[8] commons.wikimedia.org
[9] towardsdatascience.com

Here, the neural network returns a probability on each of the options. Based on this output, usually the class of highest probability is selected as decision. In this example the movie *Harry Potter and the Sorcerer's Stone* most likely belongs to the genre adventure.

The parameters we control in the neural network, are structural parameters we set in advance. Examples of this are the number of nodes in each layer, and the number of layers. The edges between nodes represent weights which determine how much they influence each other. These weights on the edges are varied by training the neural network.

3.1.2 Layers

Before we dive into the comparison of neural networks, some features of a neural network are important to know in advance: the different types of layer.

The size of the input layer is determined by the shape of the input. In our case, we have 6 dyes and 4800 pixels in each dye, giving us an input layer of 28800 nodes. The following (hidden) layers usually become bigger (more nodes) first, and smaller (less nodes) later until the desired amount of output nodes is reached. The simplest neural network contains one fully connected hidden layer, also known as a *dense layer*. In a dense layer, each node is connected to every other node in the previous layer, see Figure 12. This provides a lot of information, but also requires a lot of computation. With our input, this would lead to $28800n$ connections between the first two layers, where n is the size of the next layer.

To decrease the amount of nodes, a *pooling layer* can be applied. A pooling layer is designed to only let part of the input through. Most common in image classification are max pooling and average pooling, usually with both a stride and filter size of 2. In this example, every pooling layer divides the number of nodes by 4 in the two-dimensional case, or 8 in the three-dimensional case. The meaning of these terms is easier shown than explained. In Figure 14 both types of pooling are illustrated. A block of 4×4 becomes a block of 2×2 and no pixel is doubly used. Max pooling, as the name suggests, takes the maximum value out of each block of 4 pixels. Average pooling calculates the average of each block.

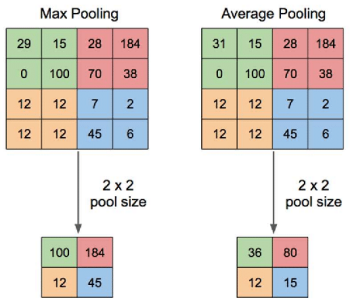


Figure 14: Max and average pooling [10]

The final type of layer we will discuss, is a bit more complex: the convolutional layer (Figure 15). This takes a rectangular part of the input (image matrix) and takes the inner product with a *kernel matrix* of the same size. A kernel matrix can be seen as a set of weights to multiply with the surrounding pixel values. The outcome of this inner product gives a single value for the middle location of the block. The block size is not necessarily 3×3, but it is a very common size to use. The goal of a convolutional layer is to use some information from a pixel's neighbourhood, but not all pixels in the entire image (like in a dense layer). If we compare this to a dense layer of 10^2 nodes, connected to all previous 10^2 nodes. A convolutional layer only has $9 \cdot 10^2$ edges, compared to the 10^4 edges of the fully connected layer. The convolutional layer is mainly used in image recognition algorithms[8].

3.2 Previous work

Now that we know the basics of a neural network, we want to study how they are currently being used in the field of bio-informatics. To the best of our knowledge, only Duncan Taylor and David Powers have published about applying neural networks to electropherograms. Because their work is a relatively novel application, we will be looking into comparing their work to other state-of-the-art machine learning techniques processing similar data. First, we look more thoroughly into the research by Duncan Taylor and David Powers in Section

[10] researchgate.net
[11] researchgate.net

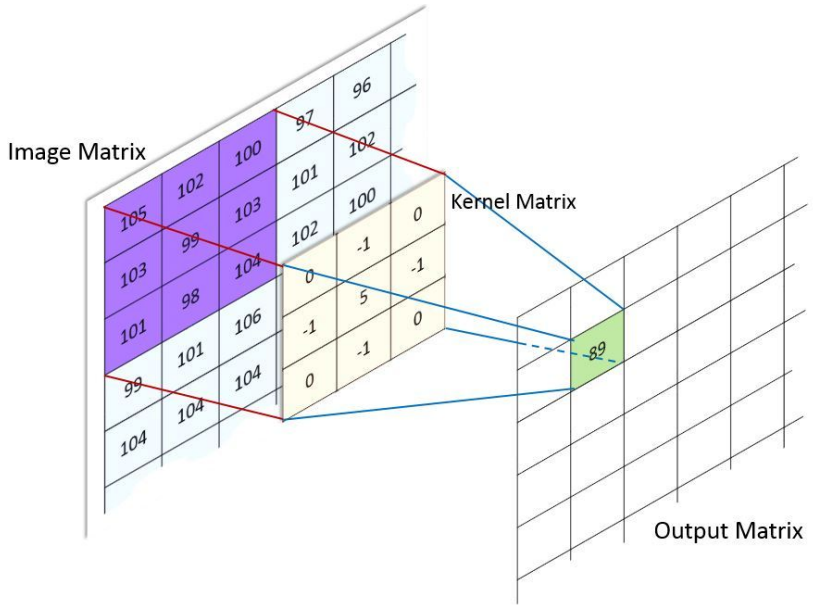


Figure 15: Illustration of a convolutional layer[11]

3.2.1. Then, we extend our search into neural networks being applied to somewhat similarly shaped biodata in Section 3.2.2. We end this literature review by concluding which neural network structure we regard best suited for this specific application (Section 3.3).

3.2.1 On applying deep learning to STR electropherograms

Our aim is to partially automate analysis of STR electropherograms using neural networks. Duncan Taylor and David Powers published a series of papers [1], [2], [3] which provide a proof of concept that neural networks perform well on electropherograms.

The first paper [1] of this series investigated whether it was at all possible to train an artificial neural network on this type of data. They consider five different labels for each pixel of the electropherogram: baseline, allele, (backward) stutter, forward stutter, and pull-up (cf Figure 7). To classify each pixel, a *window* of 100 pixels to the left and 100 pixels to the right of it in all six dyes are fed into the neural network (total 1206 input nodes). The neural network itself only consisted of the input layer, one hidden layer and an output layer; the most simple structure possible. After training on one electropherogram, the neural net already had an overall accuracy of 93% on an unseen profile (and 98% on the profile it had been trained on). For comparison, if an algorithm only predicted baseline, the most frequently occurring label, it would have an accuracy of 80%. Anything above that shows the network has learned something, but 93% is not as incredibly high as it would be for equally distributed classes. It is also important to note that training and testing was performed on a simple profile to which only one donor contributed.

Table 1: Confusion matrix from trained neural network applied to unseen profile. "Rows represents ground truth responses and columns are assigned classifications".[1]

	Baseline	Allele	Stutter	Pull-up	Forward Stutter	Error rate (fraction)
Baseline	4691	36	33	81	44	0.0397 (194/4885)
Allele	15	362	0	0	0	0.039 (15/377)
Stutter	64	0	139	0	0	0.3153 (64/203)
Pull-up	97	0	9	333	7	0.2534 (113/446)
Forward Stutter	19	0	0	0	70	0.2135 (19/89)
Totals	4886	398	181	414	121	0.0675 (405/6000)

In Table 1, the confusion matrix is shown for the neural network's prediction on the unseen profile. The baseline and allele labels were predicted correctly most often, having an error rate below 4% on a per-pixel scale. Whereas pull-up and stutter had an error rate of $20 - 30\%$, and were misclassified most frequently. These last percentages do not weigh as heavily in the overall accuracy, since the labels occur less frequently. Because the baseline label occurs the most often, the network is relatively rewarded more for correctly classifying baseline. This might explain why the error rate on the baseline is among the lowest.

Their second paper [2], written together with Ash Harrison, tries to improve the artificial neural network by using a more complex structure and more specific training. They add two hidden layers to the neural net and train on specific loci or fluorescent dyes. The complete output uses all these specifically trained neural networks to arrive at a prediction for the full profile. They also add a sixth category (half stutter) and compare the resulting data to genetic analysis software. Accuracies are quite similar to [1], but do improve a little from the extra specific training.

The third and most recent paper they wrote [3] with Michael Kitselaar, investigates the versatility of their trained neural networks. They aim to answer relevant questions such as under what circumstances another neural net should be trained, and when the neural nets can be applied to different types of data.

Throughout their papers, Duncan Taylor and David Powers mainly aimed to investigate the applicability of neural networks to electropherograms. They state in their conclusion, "Much work is required to develop and train a ANN that could be used routinely in active forensic casework; however the advantages of pursuing such a system are great." The authors believe there is definitely merit in the method, but does not compare to current standards yet.

They use two variations of artificial neural networks, and keep most other variable choices the same throughout the series of papers. There is still room for improvement by optimising each of those choices. We discussed this project work with one of the authors, Duncan Taylor, as well. A different type of neural network may be better suited for this type of data. Changing the simple neural network to a convolutional neural network (CNN) for example, since CNNs are designed to work well on images[8]. An electropherogram can be seen as a very simple type of image, and we will explore this feature in Section 3.2.2.

3.2.2 On applying deep learning to other biological data

In this section the papers are divided into a number of subsections based on their subject: basecalling, heartrate monitoring, audio signal processing and MRI. The machine learning techniques in the papers will be summarised in Section 3.2.3.

Basecalling

As we have seen in Chapter 1, DNA can be represented as a sequence of nucleotides: A, C, T and G. To find out what sequence a string of DNA consists of, in a sample fluorescent markers are added to the nucleotides. Then, just like we discussed with STRs, an electropherogram is created by measuring the fluorescent feedback for each nucleotide. Figure 16 shows an example of this.

The goal is then to decide for each position which nucleotide it is. When a trace is as clean as the one in Figure 16, it is simply choosing the nucleotide with the highest fluorescent feedback. While simple threshold-based rules to automate this process already show a relatively high accuracy in basecalling, the authors of [9], [10] and [11] felt this could and should be improved upon with machine learning.

In this paper on novel (machine learning) techniques [11], both Artificial Neural Networks and Polynomial Classifiers are used to reduce errors in basecalling of an electropherogram. They published two papers before on each of the respective machine learning techniques [9], [10]. Main problems with existing methods on basecalling are that either they are not accurate enough, too slow, or require a lot of parameters that have to be manually set. The ANN used consists only of a single hidden dense layer, and the PC consists of 3rd order polynomials. The ANN and PC require hardly any training, yet perform similarly to the current standard (ABI and PHRED). They also perform some pre-processing (color-correction, peak sharpening, normalization) to improve even further. Eventual accuracies are slightly above 98% for both the ANN and PC. This is to be expected, since the task is relatively simple. At each position, it is known a nucleotide needs to be selected. The question is just which one, out of four possibilities. What is useful to take away from this, is that machine learning methods also perform well on simpler problems.

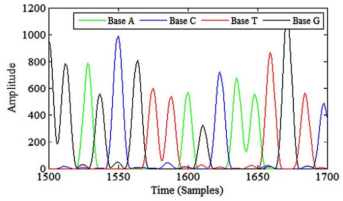


Figure 16: Segment of an SNP electropherogram trace [11]

Heartrate monitoring

Heartrates are monitored using an ECG (ElectroCardioGram) and can show many different types of aberrant heartbeat. A paper by Jimenez-Perez, Alcaine and Camara [12] applies a U-net (type of CNN) on one-dimensional heart peak data to split into the different heart waves (P, QRS and T) that make up a single heartbeat, see Figure 17.

They give a similar argumentation for using machine learning; other methods are laborious, many parameters need to be set, much knowledge and experience needed, and even then there can be different results between trained professionals. Their technique to combat the low availability of data (and at the same time overfitting) is very interesting. They add different kinds of simulated noise to the samples, based on types of noise that occur in practice. Other regularization techniques used are SDo, batch normalization, data augmentation, and semi-supervised pre-training. The accuracy percentages for different test cases are all in the 90s. They conclude that their method performs better than other machine learning methods, is more generalisable than non-ML methods, performs similarly to current state-of-the art methods, and has none of the downsides previously mentioned.

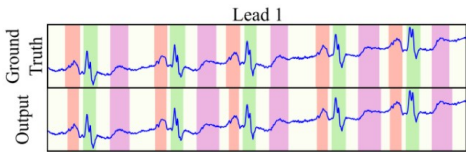


Figure 17: Delineated sample of heartbeat into P (red), QRS (green) and T (purple) waves. Top graph is ground truth and bottom graph is U-net output. [12]

In [13], Gordon and Williams use a combination of techniques (among which two ML methods) to automatically detect PVC (Premature Ventricular Contractions) in ECGs, or in easier terms: an “arrhythmic beat type”. If these occur often, it can be a precursor to a more serious heart condition. However, they occur infrequently, making manual observation impractical and expensive. Existing methods again rely on estimating a large number of parameters, and specific features which are difficult to detect. They first filter out the QRS heart wave with a non-ML technique, all further analysis is performed on the QRS wave. Main goal is to develop a more robust and general method. A convolutional autoencoder is used to combat overfitting and create a lower-dimensional feature space. Then, a random forest classifier is applied to do the actual

classification of PVC or normal heartbeat. The algorithm performed just slightly better than current, but false negatives were distinctly less; which is very useful in this specific case: it means less actual PVCs were missed. Overall accuracies are around 98%.
Another paper on the same subject is [14] by Hasan et al. It applies a one-dimensional convolutional neural network to ECGs to detect arrhythmia. The previous paper used a more advanced combination of techniques to detect one specific type of arrhythmia, whereas this paper classifies 5 different varieties with again around 98% accuracy. The neural net built from a couple of sequences of (2) convolutional, pooling and dropout layers put together, and finally followed by one flattening and one dense layer into the output layer.

Audio signal processing

The most common way to represent audio, is to create a so-called spectrogram. A spectrogram has two axes: horizontal time axis and vertical frequency axis. In the figure 18 it shows the amount of decibel in different colours. This means that audio is, in a way, three-dimensional data. This is quite a stretch from the 6 slightly correlated one-dimensional graphs an electropherogram is composed of. Since there is a lot of machine learning being applied to audio data, we decided to look into the subject regardless.

Xie et al. [15] set out to develop a new method to detect snoring by using multiple microphones. Snoring is a precursor to OSA (Obstructive Sleep Apnea), and hence very important to detect. However, not every patient has a bed partner, and not all bed partners detect snoring reliably. Current tests require patients to be monitored overnight with uncomfortable sensors, which is expensive. And waiting lists are long because of the length of such a study. The authors focus on repetitive snoring patterns, since it automatically yields a good ratio of snore to non-snore audio fragments; it is ideal to have the same amount of data in each data set. On top of that, it is easier to annotate repetitive snoring. The network uses 3 convolutional layers, each followed by a max pooling layer, the output of which is then fed into an RNN (Recurrent Neural Net), and eventually into one single dense layer. The achieved accuracy was about 95%. Interesting to note is that they see significantly more false-negatives than false-positives.

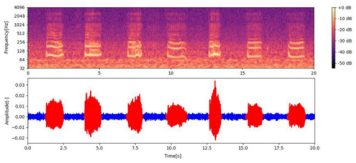


Figure 18: "Example of an audio signal containing 7 snore events. The bottom part of the figure shows an example of raw signal after applying a Butterworth HP filter, where the red parts are snore events. The top part of the figure depicts the spectrogram based on CQT." [15]

The next paper [16] by Colangelo, Battisti and Neri, applies progressive resizing (plus another standard resizing for comparison) on a pretrained CNN (ResNet34), which is then retrained on audio spectrograms. Their goal is to classify events in the spectrogram. Progressive resizing is a way to make training more efficient by increasing the resolution of the images during training. ResNet34 is pretrained on ImageNet, and uses weight decay; sets the learning to be the maximal learning rate such that loss decreases. One data set has short audio clips (less than 4 seconds long) which contain an event out of ten possibilities. The other data set has 5 second clips containing one out of fifty possible events. Overall performance does not measure up to the current state-of-the-art, but the progressive resizing itself is an improvement.

MRI

A traditional MRI scan, scans the brain in sections. A two-dimensional slice of the brain is imaged, and the scanner moves down a few millimeters to do the same. By adding all images together, an almost three-dimensional scan of the brain is obtained. You could see it as a 3D image which is discretised in one dimension.

Han and Kamdar [17] apply a bi-directional convolutional recurrent neural network (CRNN) to MRI images. It predicts *methylation status* in cancer patients. There is a gene (MGMT) whose methylation status gives information about how well chemo therapy will work. These variations of the gene, can manifest in the

size and shape of brain tumors, visible on an MRI. Recent other methods to find the characteristics usually use random forests or SVM (Support Vector Machines), which the authors of [17] disapprove of because it requires a lot of hand-curated features. So they decide to use a CRNN. The dataset is split as follows: training 70%, validation 15%, test 15%. It is a little unusual to have such a small test set, but they have a very limited amount of data. The results show only around 65% accuracy overall, but they do have about a 50/50 ratio positive/negative samples, so the network is definitely learning some features. It performs at least 10% better than the random forest classifier which is currently being used, so their research is very useful in practice. They note that their model is overfitted to the training set, and apply L2-regularization until the validation accuracy starts to drop. The neural network in this paper really suffers from the low availability of training data.

3.2.3 Summary

We have discussed research in the field of basecalling, heartrate monitoring, audio signal processing and MRI. Now, we will compare these papers to the problem at hand. We will look at the type of neural network used and how similar the problem is to ours.

Although basecalling is applied to the most similar data with respect to ours, the problem is simpler. In basecalling, it is known there should be a nucleotide at each location, it is just a matter of picking the correct one (out of four possibilities) at each step. The reading of an electropherogram first requires you to find the locations of the peaks, and then pick the correct allele out of more than four possibilities. The similarity between the data is quite obvious: both are a number of graphs (both of fluorescent feedback even) in which peaks show up signifying a certain part of DNA, although an allele is a bigger structure than a single nucleotide. The neural network structure they use, is the same as Duncan Taylor and David Powers [1] used, and performs well in this case too. This reinforces the belief that neural networks are a good direction to be searching for our desired ML method.

The second closest type of data to ours, is the heartrate data. The data is just one graph instead of six, but the problem is a little closer. The delineation of heartwaves is more like our problem of detecting peaks in the electropherogram. In the study by Jimenez-Perez, Alcaine and Camara [12], the detection is carried out by a U-net and performs quite well. A U-net is a very special type of convolutional neural net, which predicts a label for each pixel of the input image at once. Another very interesting part of this study is the addition of simulated noise to create more data and combat overfitting.
Gordon and Williams [13] and Hasan et al. [14] used a non-ML method for the most relevant part to us (the delineation of the heartbeat), and only apply the machine learning when the wave has been detected. This implies that the research by Jimenez-Perez, Alcaine and Camara provides the most meaningful results to us.

The audio and MRI data papers are both a bit further removed from the electropherogram data. Both can be seen as three-dimensional data, where our data is two-dimensional at most. The audio signal processing papers do tackle a more similar problem type: detection of phenomena (think of these as peaks). Both Xie et al. [15] and Colangelo, Battisti and Neri [16] use a type of convolutional neural network, where the first of these combines it with a recurrent layer. Colangelo, Battisti and Neri also apply an interesting technique, progressive resizing, to enhance training of the neural net. The MRI data is the most different to an electropherogram, so the research on it, [17], weighs the lightest in our comparison. One interesting fact to notice is the same type type of neural network is applied (CRNN) as for the audio detection.

3.3 Conclusion: U-net

Both the research by Duncan Taylor and David Powers [1],[2],[3] and the basecalling papers [9],[10],[11] reinforced our belief that neural networks in general could work well on electropherograms. The heartbeat delineation research [12], which is also the most comparable to our subject, pointed us in the direction of a U-net. This is a very different type of neural net, which looks at the image as a whole. It classifies each pixel of the input image, and would remove the need for the windows used by Duncan Taylor and David Powers. The audio [15],[16] and MRI [17] papers, are the most different to electropherogram data, but all

seemed to agree on using CRNNs. Some papers also provided useful techniques to combat overfitting or the low availability of data [12],[16].

From this previous research, the CRNN and U-net seem to be the best choices. We found more papers using the CRNN, but the U-net was applied to the most similar data and problem compared to ours. The U-net was selected after consulting with the deep learning experts of the FBDA team. They had previous good experience with U-nets, we saw more promise in U-nets than in the CRNNs. The U-net was also deemed as better suited for this type of data, because the entire image can be used by the neural network at once. This is confirmed by the fact that the most comparable research paper [12] also selected this structure. Not unexpectedly, because U-nets are CNNs (which tend to perform well on images) and were specifically designed for biomedical image segmentation[18]. As an additional benefit, it simplifies the input structure for the neural network as well, since the *windows* used by Duncan Taylor and David Powers can be omitted. In the next section, the structure of the selected U-net will be discussed in more detail.

4 U-net

This chapter is dedicated to information regarding the U-net. We argued this choice in the previous Section 3.3. In Section 4.1 the concept of a U-net will be explained. Then, in Section 4.2, the implementation and design of the U-net for this specific project will be discussed. Section 4.3 describes the format of the input and labels that go into the U-net.

4.1 Structure

A U-net[19] is a specific type of CNN (convolutional neural network), which is very logically named: it is shaped in the form of a U. Where most neural networks are structured to predict one label for one image, a U-net predicts a label for each pixel in the input image.

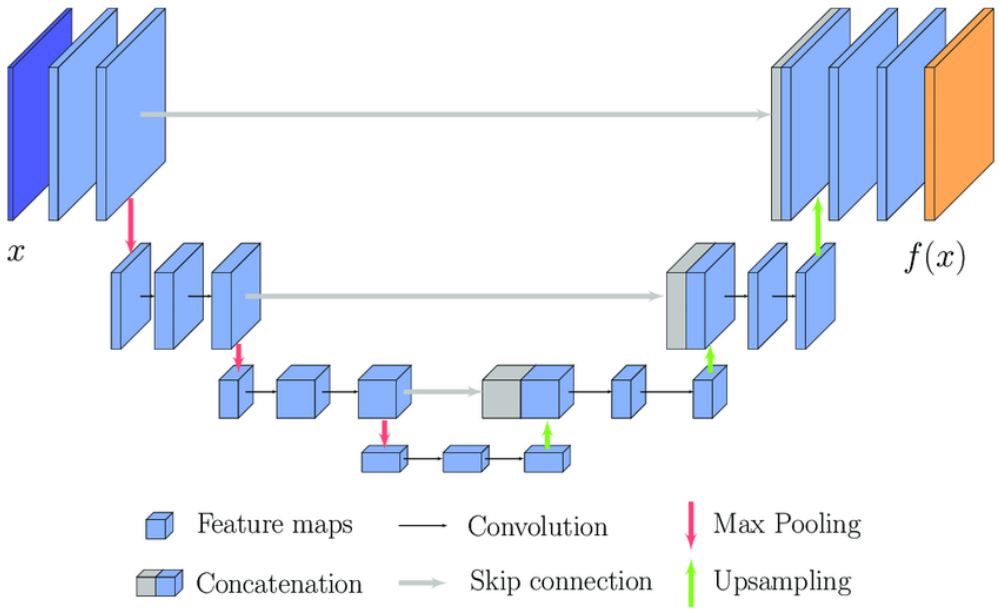


Figure 19: Example of a U-net [12]

The first step in the U-net is a convolutional layer, as was explained in Section 3.1.2. It creates a lot of *feature maps* by applying different filters on the input image, to try and find useful information. A feature map can be seen as a representation of the input. Feature maps can have emphasis on areas of high value, or on edges, or on specific shapes even. As more convolutional layers are applied, information from increasingly further away in the image can begin to impact a pixel. To demonstrate this, Zeiler and Fergus [20] engineered a way to visualise the intermediate feature maps in their network.

[12] researchgate.net

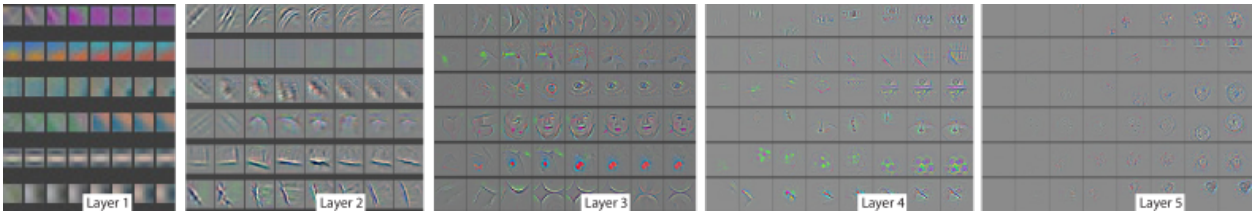


Figure 20: "Evolution of a randomly chosen subset of model features through training. Each layer's features are displayed in a different block. Within each block, we show a randomly chosen subset of features at epochs [1,2,5,10,20,30,40,64]. The visualization shows the strongest activation (across all training examples) for a given feature map, projected down to pixel space using our deconvnet approach" [20]

After looking closely at the different layer's feature maps in Figure 20, you might guess that the images show people's faces. The first layer identifies lines, the second layer finds longer lines or even curves, and the third layer already shows parts of the faces. They could have set the kernel size to be larger to start with of course. However, by choosing a small convolutional kernel size, and adding multiple convolutional layers, the same (or even more) information is obtained more (computationally) efficiently.

This convolutional step is visualised in Figure 19 by the input image (dark blue) getting more depth in two steps (light blue). The figure shows two convolutional layers were applied (although the arrows are not visible in the top left yet). The current result contains (on purpose) too much information, instead of only the important parts. So, the result is fed to a (max) pooling layer (red arrow down), to reduce the amount of information. These steps are repeated a few times (three in this example in Figure 19). All these operations together form the *down* path of the U-net.

By the time you have arrived at the bottom of the U, the height and width of the image are a lot smaller, but the depth has only increased. The network has created a lot of feature maps, but the other dimensions were limited. The idea is that this forces the network to focus on the most important features, and not on their precise location. Now on the *up* path of the U, we want to do the opposite; try to limit the feature maps, and recreate the original size of the input. We want to retrieve some locational information and apply the discovered useful features.

The intermediate result now has to be *upsampled*: increasing the resolution (height and width) of the image (green arrow up). Each pixel is simply repeated a few times (with the same size as the pooling). This upsampled result is concatenated with the corresponding same-size layer on the down path (see the horizontal grey arrow and block in Figure 19). The idea is that this horizontal skip connection provides the needed locational information to return to a higher resolution image. The concatenation is along the filter axis. This is again fed into two convolutional layers. The upsampling, concatenating and convolutional steps are repeated a number of times (the same number as on the *down* path), until the original size of the input is achieved again.

4.2 Settings

Now that the general idea of a U-net has been introduced, we explain more details about the specific settings chosen in our implementation. Each block of layers (two convolutional and one pooling layer) is repeated three times. On the up path, their equivalent blocks (upsampling, concatenation and two convolutional layers) are also repeated three times. This shape of U-net is the same as Figure 19 shows. All coding is done using Python 3.8[13] using the PyCharm IDE[14].

[13] python.org
[14] jetbrains.com/pycharm

4.2.1 Convolutional layers

The first layer, besides the input layer, in the U-net is a convolutional layer. The padding is set to "same", so the output has the same shape as the input of this layer. We opted for a kernel of shape 3×6, so all dyes can add information to each pixel. The 3 is a very common choice, the 6 is specifically chosen for our input shape. In general, these parameters can be kept very small, since layers are applied multiple times. The information throughout the entire image can incrementally influence more pixels while traveling down the U. First, pixels are only influenced by neighbours, then by neighbours of neighbours, and so on.

The convolutional layer has a 2×6 kernel on the up path, because the third place is taken by the concatenated extra maps. Furthermore, we use ReLU activation ($\max(input, 0)$) and the He normal kernel initialiser [21]. These settings are commonly used in neural networks as they perform well and robustly. Each set of two convolutional layers, is set to double the amount of feature maps in the previous layer.

4.2.2 Pooling layers

The first important setting of the pooling layer, is that we decided to use an average pool, instead of the more common max pool. We hypothesised an average pool is more susceptible to subtle changes in the data and shape of the graphs, which could be useful on the up path of the U-net. Section 6.2 will delve deeper into the effects this change had on the U-net.

Another setting we changed, is that pooling layers never pool in the direction of the dyes. More specifically; the pooling layers use a shape of 2×1. We do not want to lose too much information. The 6 graphs per dye are only very loosely connected to each other, and we did not feel it would be effective to try and reconstruct the dyes' graphs from the neighbouring dyes' graphs. The information from the dyes can already influence the other dyes, because we changed the convolutional kernel to a size of 3×6 (cf Section 4.2.1). By pooling in the dyes' dimension, we would be left with only the average of two dyes, and eventually with only the average of all dyes. It would be very hard for the network to predict accurate labels for each pixel in each dye from only this average and the skip connections.

We pool with a stride of 2. This means we skip every other pixel (only pool on every second pixel), and it cuts the result's dimension in half with each pooling layer. It is almost exactly as the example in Figure 14 in Section 3.1.2, except that we only pool in one of the two dimensions.

4.2.3 Hyperparameters

Finally, there are some global settings that influence the training of the U-net: hyperparameters. As we have discussed in Chapter 3, a neural network starts off quite random, and needs time and data to learn from its mistakes. Hyperparameters are a way to finetune this learning process.

One run through all data is called an *epoch*, and we train the network for 100 epochs. We achieved this number by storing the intermediate weights to repeatedly initialize new runs to keep on training. After 100 epochs the U-net was done training, as more training did not have any significant effect on the loss or metric. The learning is performed in batches: we set the number of inputs that are processed before updating the U-net to 10 (*batch size*). A small batch size allows the network to train on specific images and find subtler details, but might lead to overfitting. A larger batch size makes the network see more images before updating its weights and is more generalisable to the whole data set.

Then, we logically need some measure to translate the performance into a value. A neural network tries to optimise a so-called loss function. We used binary cross entropy as our loss function.[15]. To move into the direction where our loss is the most optimal, an *optimizer* is employed. We used the Adam optimizer with a default learning rate of 0.001.

[15] tensorflow.org

4.3 Input

The electropherogram as an input is two-dimensional; six one-dimensional graphs. This is in contrast to the general structure of a U-net, the example in Figure 19 shows three-dimensional input. The input is cropped to be 4800×6 so it is easily divisible by the pooling layers of the U-net. This cropping is necessary, because the length of the graphs differs for each electropherogram, and a U-net requires the inputs to be of the same size. Each basepair is 10 pixels wide in the electropherogram, so the 4800 pixels left after cropping constitute 480 basepairs. The data is only cut off horizontally, so highest and lowest fragment lengths in the electropherogram are removed. This does not cut off any loci or alleles, since we only remove the first 50 basepairs where there are no loci. And the data ends at the 530th basepair, after which there are no more loci. In Figure 21 the left and right cutting lines are shown for one dye.

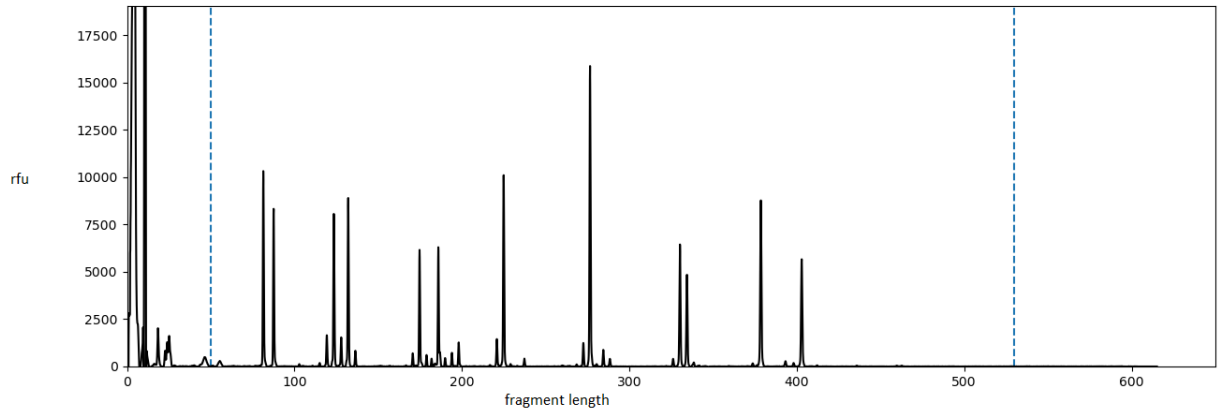


Figure 21: Example graph of electropherogram with left and right cut off points (dashed blue line)

Another operation that we apply on the input, before feeding it into the neural network, but after the cutoff, is rescaling. The values of an electropherogram (in rfu) can reach more than 10000. By rescaling the values between zero and one, the U-net can train a lot faster and more efficiently. In a way, you are telling the neural network there is not a lot of (important) difference between a value of 19999 and 20000.

The rescaling is specifically applied after the cutoff, to remove strong artefacts often encountered in the first few pixels (*primer dimer* cf Section 2.2.1). See, as an example, the large peaks to the left of the first dashed blue line in Figure 21. These are all caused by primer dimer. Each image is rescaled separately, since the quality of a sample can influence the height of peaks. It is rescaled for the entire image (all six dyes) at once, because pull-up heights would be severely distorted if we rescaled each dye's graph separately. After these two operations, the cropped, rescaled electropherogram is fed into the neural network.

4.3.1 Label format

In Section 2.3, it was explained how the per-pixel labels were obtained. We are using the theoretical expected peaks as ground truth labels. In addition, and as mentioned in Section 2.2, we also have access to an analyst's annotated peaks for each electropherogram. An advantage of the latter is that we can compare the results of our U-net against human performance. Labels are entered into a two-dimensional array, where a zero means no peak and a one means it is denoted a peak. To match our input shape, the labels are also a two-dimensional 4800×6 (boolean) array.

5 Evaluation

Before we can discuss the results in Chapter 6, we need to decide how to handle and evaluate them. Section 5.1 describes the algorithm we designed to call alleles from the U-net output, and Section 5.2 how we calculated an upper bound on this algorithm. Section 5.3 covers how we scored this output in terms of alleles.

5.1 Allele calling algorithm

The eventual goal of the U-net, is to predict which alleles are present in an electropherogram. So we need an algorithm to go from the predicted per-pixel labels back to the alleles. We do have the information of each allele's approximate location on the horizontal fragment length axis. We use the same information from the Genemarker™ panel file, previously discussed in Section 2.3 when it was used to create the per-pixel labels. There are some challenges with this problem, which we will tackle one by one.

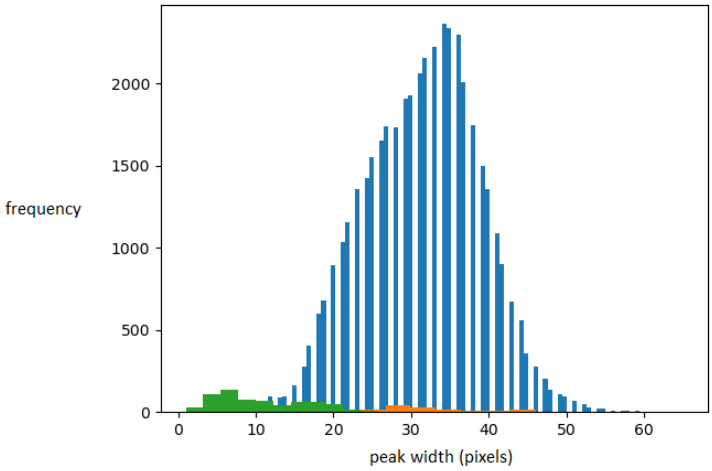


Figure 22: Histogram of widths of regions designated as peak by the peak selection algorithm from Section 2.3.1. Green: peaks with maximum height below 100 rfu. Orange: peaks selected by 2 or more alleles. Blue: all other peaks.

The allele bins given in the Genemarker panel file are approximately one basepair wide, but the average peak width is larger than that. In Figure 22, we plotted the result of our peak selection algorithm. It shows the width of a peak (in pixels) on the horizontal axis, and the number of times this peak width occurred on the vertical axis. Remember that a width of 30 pixels corresponds to a fragment length of 3 basepairs. Figure 22 shows a, somewhat Gaussian-looking, distribution centered around a value between 30 and 40 pixels, or three to four nucleotides.

Because some alleles in the mixture, may not appear in the electropherogram (dropout[16]), we also plotted which peaks were extremely low (below 100 rfu) in green. In orange, we plotted the peaks that were selected by more than one allele bin by our algorithm described in Section 2.3. This means that the resulting peak was either a combination of two smaller peaks, or one of the peaks dropped out.

Knowing the disconnect between information (allele bins of one nucleotide wide) and the U-net output (full peaks of three to four nucleotides wide), we designed a simple algorithm to translate the output. The first step is to round off the U-net's output to a value of either zero or one. The threshold is set at 0.5. Then, for each block of ones (a peak), we find the most probably correct allele bin. In each block of positive prediction,

[16] sciencedirect.com

we check which bins are in there, and how high the (unrounded) U-net output is for all pixels in the bin. All these probabilities are multiplied together to form the probability of that allele causing the peak. The allele bin of highest probability is chosen as label for that peak.

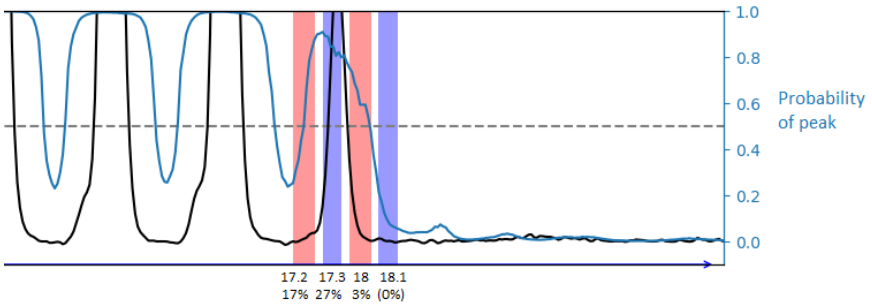


Figure 23: Visualisation of allele calling algorithm. Background shows allele bins in alternating colours. Blue graph is U-net output, black line is electropherogram trace for reference. Grey dashed line is U-net output value of 0.5. Numbers at horizontal axis show alleles and corresponding probabilities for a single peak.

Figure 23 shows an example of the algorithm's working. We look at a peak where the U-net is more uncertain of its prediction. All closest allele bins are plotted as background, in alternating colours to be easily distinguishable: 17.2, 17.3, 18 and 18.1. Allele 18.1's U-net output is never above 0.5, so is not considered as label. The algorithm considers three bins, and only the right part of the 17.2 bin above 0.5. The U-net output values in these bins are multiplied together for each bin, resulting in the probabilities given in Figure 23. For this peak, the algorithm will label it as allele 17.3.

5.2 Upper bound algorithm

The need for this *upper bound algorithm* will be described later in Section 6.4. The upper bound algorithm provides an upper bound on the performance of the U-net with the most optimal allele calling algorithm. This section describes the upper bound algorithm itself using an example in Figure 24.

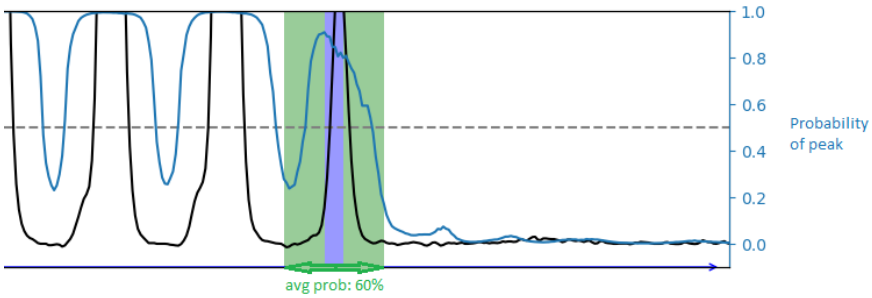


Figure 24: Visualisation of upper bound algorithm. Black line is electropherogram, blue line is U-net output. Blue background is the allele bin, green background area to be checked if average probability is over 0.5 (grey dashed line).

To calculate this upper bound, we use the ground truth. We check for each allele in the ground truth, whether the average probability is high enough (i.e. over 0.5) on the expected resulting peak, see Figure 24. We plotted the same part of the same electropherogram as in Figure 23, but only show the (correct) 17.3 bin and disregard the other allele bins. We regard the expected peak as the pixels in the allele bin (blue background in Figure 24) $\pm$ 15 pixels to its left and right (green background in Figure 24). This corresponds to a peak width of about 3 to 4 basepairs wide: the allele bin has a width of 8-10 pixels, and we add twice 15 pixels. In Figure 24 the expected peak window to be checked is the complete green plus blue background.

The blue line shows the U-net output, which we use to calculate the average probability on that interval.

The allele bins may be shifted from the center of the peak, so the expected peak window may not perfectly contain the peak, but the average should fix this problem. If the peak is detected by the U-net, all output values on the peak are close to a value of one. If part of this expected peak window has lower values, the average should still be well over 0.5, as it is in the example in Figure 24 too.

If, according to this check, the probability is high enough, we add this allele to our list of predicted alleles. Once we have checked all alleles in the ground truth (positives), we set the U-net output in those locations to zero. After this first part is done, we run the previous algorithm from Section 5.1 to see if the U-net has detected any more (wrong) peaks (false positives).

5.3 F1-score

To compare our U-net's results to the performance of a trained analyst, we calculate the F1-score[22]. While accuracy (correct/total) gives a good idea of the total number of pixels classified correctly, it is less useful if we are mainly interested in the alleles. To explain the F1-score, it is important to understand the concepts true/false positives/negatives.

After receiving the output of the U-net, and translating this to alleles, we have two lists of alleles present in the mixture: ground truth and predicted. If the two overlap, it is called a *true* positive/negative (see also Figure 25). If the predicted and ground truth labels do not agree, it is called a *false* positive/negative (see also Figure 25. For each possible allele, it can either be present (positive) or not present (negative) in the mixture. If AMEL X was predicted, and is correct, this is an example of a true positive. If AMEL Y was predicted, but is not present according to the ground truth, this is called a false positive: it was falsely classified as a positive label. The same goes the other way around for false negatives. We can now express the accuracy in terms of true/false positives/negatives. The accuracy is the true positives plus the true negatives divided by the total.

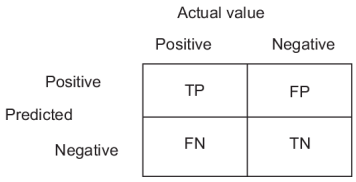


Figure 25: "Classification of a prediction into True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN)."[23]

On to the F1-score: the harmonic mean of precision and recall. Precision is the true positives divided by the total predicted positives (true plus false positives). Precision is a measure of how accurate your prediction of a positive label is. Precision is optimised when there are no false positives, and it has a value of 1. Recall is the true positives divided by the total ground truth positives (true positives plus false negatives). Recall is a measure of how many of the positives your algorithm misses. Recall is optimised when there are no false negatives, and it has a value of 1. The F-score can be tuned to put more emphasis on either precision or recall, but the F1-score is precisely in the middle: the harmonic mean[17] (a specific kind of average). This again has an optimal value of 1 for a perfect algorithm. Now that we have introduced the score we are using, we can start to interpret our results.

[17] en.wikipedia.org

6 Results

First, we will look at the output of the U-net in detail in Section 6.1. Based on this output, we will explain why the switch from a max pooling to an average pooling layer was made in Section 6.2. In Section 6.3 we will compare these results to an analyst's reading of the electropherograms. Then, we will show an upper bound on the algorithm in Section 6.4, and evaluate the performance of this final result in Section 6.5.

6.1 Interpretation

The output of the U-net is the same shape as the input labels. It can output values between zero and one. During training, the (binary cross entropy) loss function is calculated on the output values rounded to a zero or a one. For easier reading, the output value of 0.5 is also plotted in Figure 26 as a grey dashed line. We also plot the ground truth labels as background for the input (black) and output (magenta). An output value of one corresponds to a prediction of an allelic (or size standard) peak, and a zero to background. So we can interpret each pixel's output value as a probability of an allele being present at that pixel. The overall performance on a per-pixel scale of the U-net was around 95%. The example in Figure 26 is representative for the full data set. Figure 26 demonstrates the performance of the U-net on a relatively complicated mixture profile. This profile was composed by mixing 4 donors in a ratio of 20:1:2:1. So we can expect a large variety in (allelic) peak heights in this electropherogram.

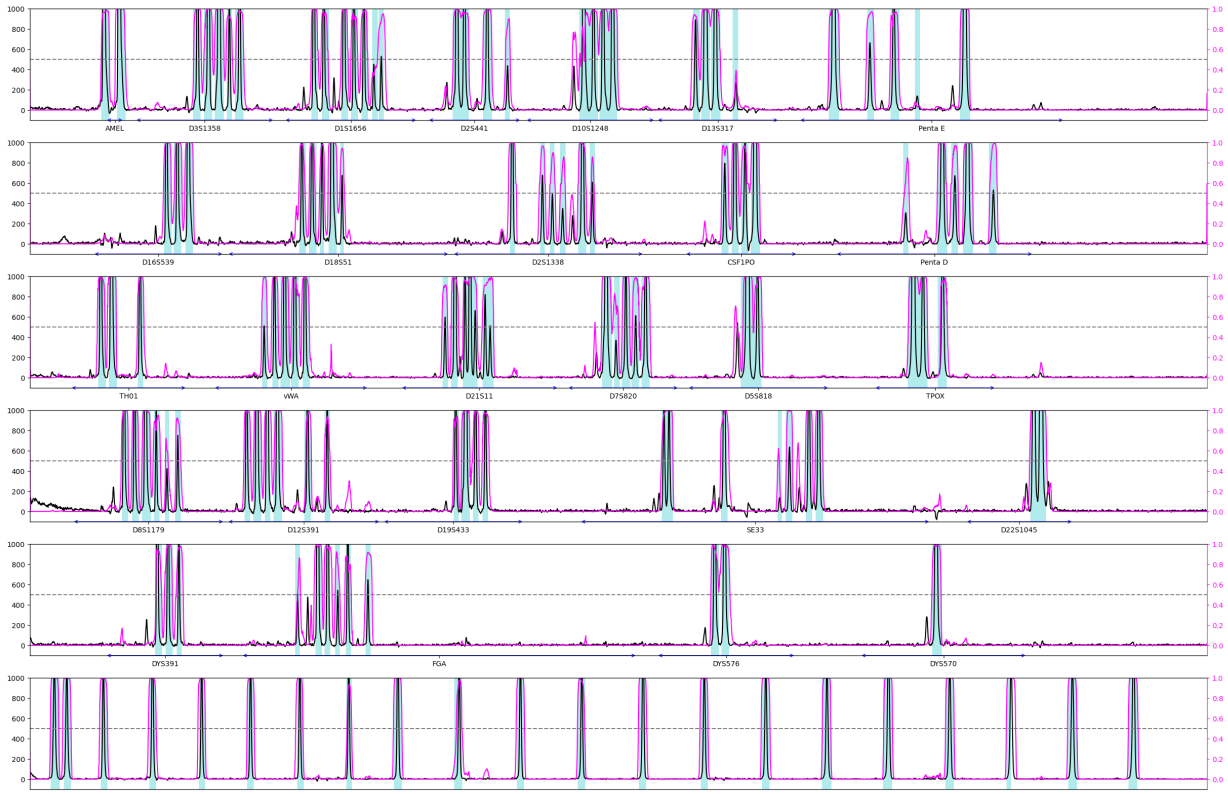


Figure 26: Example output of U-net. Background colour shows ground truth labels of allelic peaks in cyan. Black graph is input (left axis), magenta graph is U-net output (right axis), grey dashed line signifies output value of 0.5. Text and arrows on axis of each graphs are names and ranges of loci respectively.

We can see that the U-net output (magenta line) tends to predict a value close to one whenever an allele is present (cyan background). And it predicts a value close to zero when there is no allelic peak (white/no background colour). This is exactly what we wanted to happen. Upon inspecting the result of the U-net a little closer, it seems to generally follow the input graph somewhat. This is what we expect: a high probability of allelic peak on high values of rfu. The difficult decisions arise with the lower peaks that could be

either artefactual or allelic. Note also that the high values of rfu are not shown in the graph, only up to 1000 rfu. This allows us to see more detail around the baseline and small artefacts.

The U-net does make some mistakes; see locus D1S1656 in the top graph in Figure 26. Remember that the output is rounded off to a positive label (allelic peak) when it is above 0.5 (grey dashed line). The last two peaks on this locus are both allelic, but the U-net only predicts the last one, and not the second-to-last peak to be allelic. A few loci later, on D10S1248, the first peak is artefactual, yet the U-net predicts that it is an allelic peak. So the U-net makes mistakes both ways, although generally it clearly predicts a very high probability (near one) on actual peaks and a very low probability on background (near zero). The size standard (bottom graph in Figure 26) is predicted almost perfectly. Not entirely unexpected, as this easiest of the six graphs with hardly any varieties in peak heights, shapes, widths or distances.

6.2 Max to average pool

Now that we are familiar with the output of our U-net, we want to substantiate our decision for the average pooling from Section 4.2.2. Below, in Figure 27 we plotted the same electropherogram as in Figure 26, only with all pooling layers in the U-net set to a max pool instead of an average pool. Networks were trained for the same number of epochs. This figure is representative for the performance of the max pooling network on the full data set, as are the comments in the rest of this section.

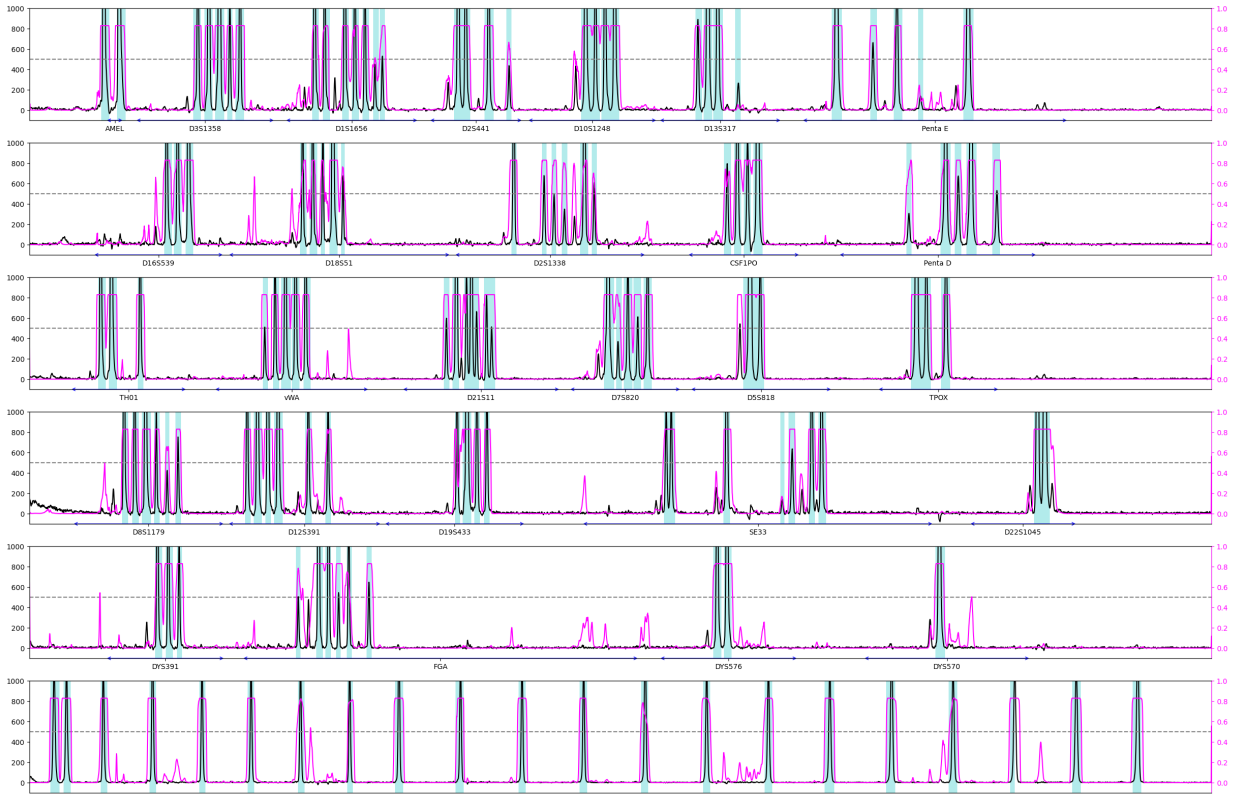


Figure 27: Example output of U-net using max pool on same profile as Figure 26 (see this figure for explanation of colours used)

When comparing Figure 27 to Figure 26, we immediately notice that the U-net output (magenta line) does not reach as high as in the average pool case. The output of the max pooling network starts off at 0.5 before training, and slowly becomes more extreme, moving values closer to zero and one as the training continues. If

we had let our network continue training, the highest probabilities would have moved closer to the top value of one. The average pooling network takes less epochs to finish training, where the max pooling network needs more than 100 epochs to be certain enough of its predictions.

Another difference between the max pooling (Figure 27) and average pooling (Figure 26), is that the max pooling network shows more random spikes around zero. Take a look at the size standard (bottom graph), and remember that the average pooling network predicted this almost perfectly. The max pooling network's output shows random spikes between the actual peaks of the size standard. There is even a spot where the spikes pass the important value of 0.5. And even in this easy-to-predict size standard, the outputs still never quite reach a maximum value of one.

Unexpectedly, the exchange of max for average pooling, also sped up the training of the neural network. Seeing it outperform the max-pooling network in terms of speed, without showing impairment on the quality of the output, we decided to use the average pooling variant of the U-net going forward.

6.3 Analyst

From the analyst, we have a list of annotated alleles for each sample. In Section 5.1 we have explained how we obtain a list of alleles based on the output of the U-net. To compare their performance, we have plotted side-by-side boxplots of their F1-scores in Figure 28. The green bar shows the median, and the blue box contains the 25% to 75% area of scores. The so-called whiskers (black bars at the end of the blue lines from the boxes) signify the minimum and maximum (excluding outliers). The outliers are represented by black circles.

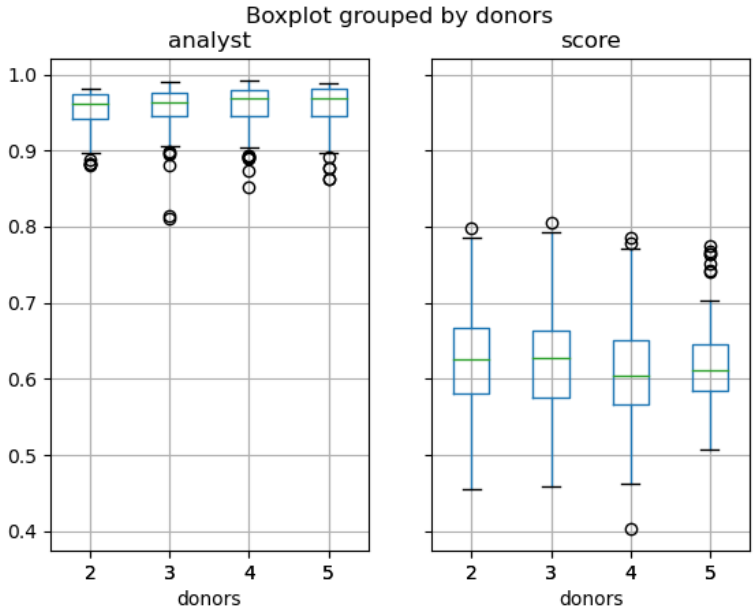


Figure 28: Boxplot showing F1-scores of analyst and allele calling algorithm's annotated alleles, grouped by the number of donors

We have grouped the results by the number of donors that contributed to the sample (horizontal axis). We were expecting a very clear negative correlation between the F1-score and the number of donors, because profile complexity increases with the number of donors. However, both the U-net output and analyst perform similarly across the different numbers of donors. What stands out the most, is that the U-net does not come close to the analyst's performance. Where the U-net has median F1-scores of slightly over 0.6, the analyst performs incredibly well with a median score of over 0.95. There are some F1-scores for the U-net around

0.8, which is almost equal to the worst performance of the analyst.

Another interesting observation, is that there is less variation in the U-net's F1-scores for 5 donors than for the smaller numbers of donors. As stated previously, we would expect the 5 donor-profiles to be the most difficult. The median score is similar, but both bars representing the top and bottom 25%, and the whiskers, are much closer to the median. This shows us that the F1-scores are more clustered together.

6.4 Upper bound

After seeing these results, we concluded that the current pixel to allele translation does not perform well enough. With a 95% accuracy of per-pixel labels, we would expect better F1-scores than a median of 0.65. Many mistakes in this translation are made due to the wrong allele being called, after the correct peak was found in the per-pixel data. Alleles are not only distinguished by the number of (complete) STRs, but also if they have a part of the sequence extra. This is denoted by e.g. 13.1, 13.2 etc. for each extra basepair following the sequence. If the repeating sequence is ATC for example, a 3.1 would be ATCAT-CATC**A**. The allele bins are very close together, because STRs with one basepair difference are very close together on the horizontal axis in the electropherograms. Remember that a basepair spans 10 pixels horizontally, but the peaks are generally around 30-40 pixels wide (Figure 22) and span more than one allele bin.

When creating the per-pixel labels in Section 2.3, we ran into some similar problems because of the allele bins. Some allele bins only intersect the side of the peak they cause, and will not be called correctly by our algorithm. Our algorithm is designed to select the most probable allele. Since the U-net output drops to zero at the sides of a peak, an allele bin that only intersects the side of a peak will never be most probable.

These problems are mainly caused by a small disconnect between the theoretical allele bins, and the actual resulting peaks. Genemarker™ solves this problem by using a ladder (cf Section 1.2.3). To solve our problem, we would need to mimic the analysis performed by Genemarker™ . However, this is quite advanced and difficult analysis, and requires more extensive research and more time than we had left at this point. Because we did want to estimate how well the U-net could potentially perform if the translation were perfected, we decided to calculate an upper bound on the score (cf Section 5.2): given that we have the best possible allele calling algorithm, how well does the U-net score? This will allow us to calculate an upper bound on the F1-score of the U-net output, which we can compare to the performance of the analyst.

6.5 Comparing upper bound and analyst

First, we begin by comparing the upper bound score for the U-net from Section 5.2 to the analyst. A boxplot of these scores, again grouped by the number of donors, can be found in Figure 29.

The most notable difference is the huge improvement of the U-net's score. Where we saw a maximum of 0.8 and medians around 0.65 in Figure 28, the upper bound has maxima nearing one and medians around 0.95 in Figure 29. Interestingly, the U-net seems to have the most difficulty with the two-donor samples, and appears to perform better for more donors. We cannot explain this phenomenon, but we do have a hypothesis. The number of distinct alleles per locus increases (relatively) less and less as more donors are added to a mixture, because five donors are more likely to have some alleles in common than two donors. It could be that the network has learned to expect a certain number of distinct alleles per locus, and cannot find enough peaks in the two-donor samples.

The F1-scores for three, four and five donors are similar to the analyst's F1-scores; medians well above 0.95 and not too much variation. However, only the U-net has a median F1-score between 0.9 and 0.95 for the two-donor group, and the longest whiskers. This is the only group, both for the U-net and analyst, where the median F1-score drops below 0.95. Comparing the boxplots of the U-net to the analyst, the U-net's scores generally seem to be more clustered together. Both the bars for the 25th and 75th percent (box) and the bars for the minima and maxima (whiskers) are closer to the median (green bar) than for the analyst. The difficulty the U-net has with two-donor samples, is not visible for the analyst. The scores for three, four

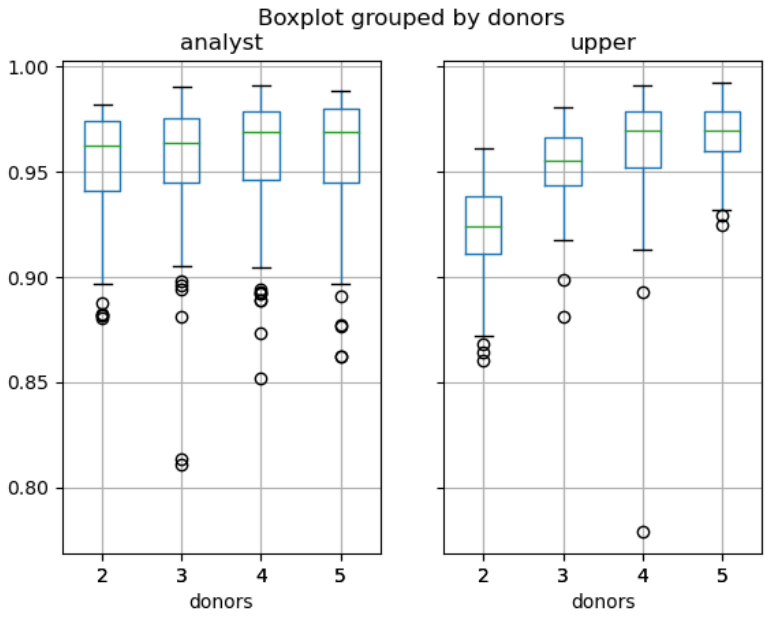


Figure 29: Boxplot showing F1-scores of analyst and upper bound algorithm's annotated alleles, grouped by the number of donors

and five donors are very similar for the analyst and U-net. All in all, performance is comparable between the two options to find alleles in an electropherogram.

Now that we had a score we could compare to the analyst in a meaningful way, we wanted to see if we could group the scores by a more interesting condition. The number of donors that contributed to a sample is not the only factor that determines the profile's complexity. Another, or maybe even more impactful factor, is the ratio between the donors. This tells us whether all donors contributed the same amount of DNA, or whether there was a clear major donor who contributed the most, and how little the minor donors contributed. For a complete overview of the ratio's between donors in the dataset, see Figure 30.

Mixture Type	**Number of contributors**			
	2	3	4	5
	Picograms DNA per contributor			
A: major 2x more than any minor	300:150	300:150:150	300:150:150:150	300:150:150:150:150
B: major 10x more than any minor	300:30	300:30:30	300:30:30:30	300:30:30:30:30
C: 2 majors with equal amount	150:150	150:150:60	150:150:60:60	150:150:60:60:60
D: major 5 to 2.5x more than minors	150:30	150:30:60	150:30:60:30	150:30:60:30:30
E: major 20 to 10x more than minors	600:30	600:30:60	600:30:60:30	600:30:60:30:30
Number of mixtures	5	5	5	5

Figure 30: "Mixture proportions and amounts of DNA used per donor to create a total of 20 different mixtures per dataset."[6]

The mixture types vary from a 1:1 to a 1:20 ratio between donors.
We would expect the algorithm and analyst to have the most difficulty with the latter, and the least with the more equal ratios. When some donors contribute a relatively small amount of DNA, that causes smaller peaks. This makes it harder to distinguish between allelic and artefactual peaks based on height.

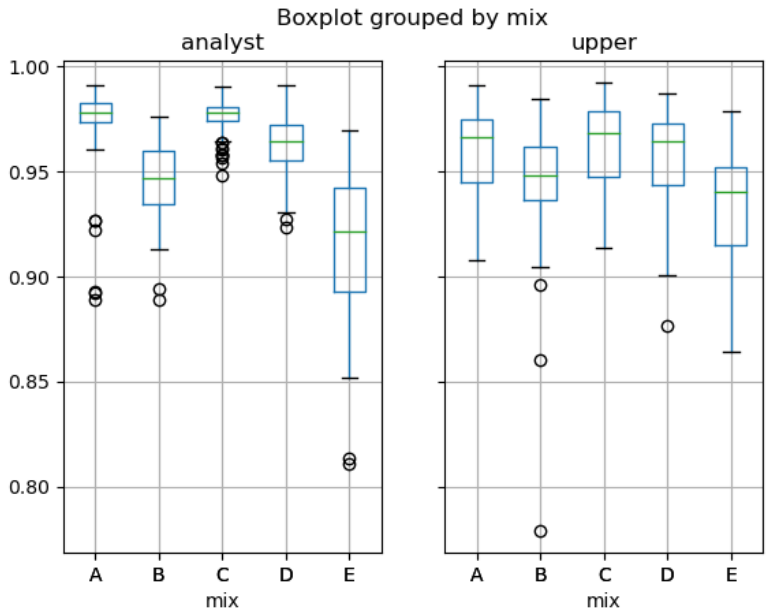


Figure 31: Boxplot showing F1-scores of analyst and upper bound algorithm's annotated alleles, grouped by mixture type

Comparing the table in Figure 30 and the boxplots in Figure 31, we can definitely see a correlation. The simplest mixture types in this data set, A and C, with ratios of 1:1 and 1:2 clearly show the highest F1-scores. For the analyst this effect is the most dramatic: mixtures A and C not only have a higher median score, but all scores are clustered close together around the median of ∼0.975. The minima are above 0.95 even. For the upper bound on the U-net's output, this effect is less dramatic, but still visible. Mixtures A and C show the highest median scores, but not necessarily the most clustered scores. For the U-net, all scores are clustered in a similar way (approximately the same size boxes and whiskers). The minima of mixtures A and C are similar to those of B and D, which are a step more complex.

After the mixtures A and C, we can see that both the U-net and analyst show the same trend. The scores become lower from A and C to D-type mixtures, then B-type and finally E-type mixtures. This was to be expected, since this is precisely the ordering of 1:1 to 1:20 ratio of the donor's contributions. In the analyst's scores, we see another trend: the scores become more scattered (the boxes become larger). This effect is most clearly visible for mixture type E, which also attains the lowest minimum for the analyst. This scattering is partly due to some scores still being quite high, while others are becoming worse.

The analyst still attains the same maxima for mixture types A, C and D (around 0.99), but drops to around 0.97 for mixtures B and E. For the U-net's scores, the maxima decrease more subtly, but they do clearly decrease for more complex mixture types. All in all, the F1-scores behaved like we expected them to: they decreased as the mixtures became more complex.

Finally, we decided to also group the scores based on the donor set. The donors were divided into six groups. We did not expect there to be any difference across these groups, so it provides a check on the validity of the results. The first donor set is made up of high allele sharing donors, the second of low allele sharing donors, and the remainder is random. This should not impact the allele calling, it just means there will be more distinct peaks in the low allele sharing donor set, and less in the high allele sharing donor set.

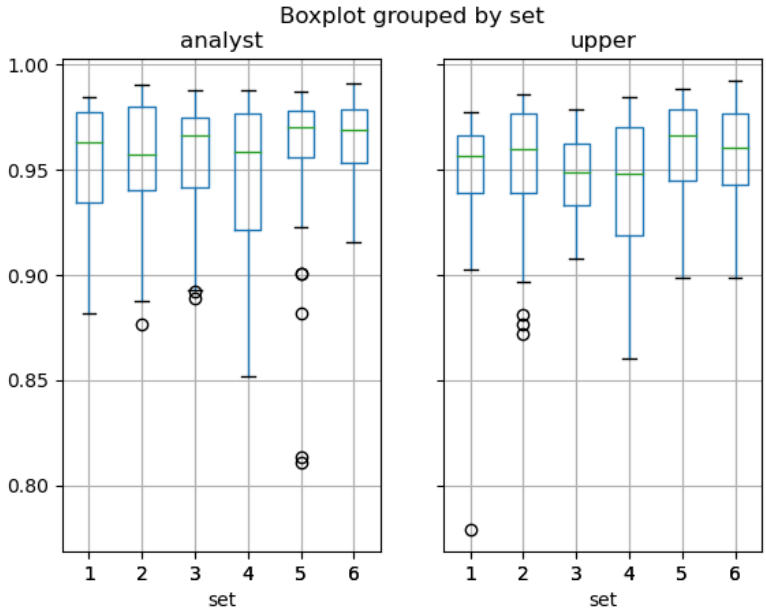


Figure 32: Boxplot showing F1-scores of analyst and upper bound algorithm's annotated alleles, grouped by donor set

The analyst performs comparably across all donor sets. Median scores for donor set 2 and 4 seem slightly lower. Most notable is the low whisker for donor set 4. However, it is important to remember the left vertical axis' scale. All these scores are already incredibly high, so small variations are magnified in this plot. Based on this boxplot, we do not see any reason to believe there is a difference in performance by the analyst across the donor sets.

Now, looking at the U-net's upper bound score, we see similar F1-scores compared to the analyst. There is no sign that the donor sets impact the F1-score in a major way. Strange enough, the upper bound also has its lowest whisker at the fourth donor set, just like the analyst. The medians seem quite constant across all donor sets. They are quite comparable to the analyst, but do not vary in the same way for each donor sets. Where the analyst's median is a little lower in donor set 2, compared to donor set 1 and 3, the upper bound score's median is actually a little higher. This just goes on to show that these are very different methods to arrive at the same kind of solution.

6.6 Summary

While the U-net's performance on a per-pixel level was good (95% accuracy), there was some trouble translating this output to alleles. We developed a simple algorithm to this end, and calculated F1-scores on the alleles predicted by the U-net and analyst. This simple algorithm to call alleles, used only the U-net output and allele bins, and scored decently (∼0.65 F1-score), but did not compare to an analyst's performance (∼0.95 F1-score). We concluded that we needed more advanced analysis (like Genemarker™ 's use of the ladder) to correctly translate the output's peaks to alleles. To give an idea of the potential of this method, an upper bound was calculated. This upper bound's scores were comparable to the scores of an analyst.

7 Conclusion

In Chapter 6, we discussed the U-net's output in depth. We compared the U-net to the analyst's performance in two ways: both as is, and with an upper bound on the score. We will revisit our results to conclude that the method of applying a U-net of per-pixel labels of an electropherogram definitely shows potential.

Based on the performance on a per-pixel level, the U-net achieved a roughly 95% accuracy. By normalising the input data and using average pooling layers, the training sped up and this accuracy was achieved after 100 epochs. Looking at only this result, we would say the U-net was successful.

When translating these per-pixel labels back to alleles present in the sample, we ran into some problems. We use information from the Genemarker™ panel file to locate allele bins on the horizontal axes. However, Genemarker™ applies more advanced analysis to call the alleles correctly (namely a ladder). In our case, this preprocessing has not been applied yet, since we opted for a more raw data type for the electropherograms. The naive algorithm calling alleles from the U-net output, achieves only about a 0.65 F1-score compared to the ground truth alleles. Although a value of 1 might be unattainable because of possible dropout, we were not satisfied with this score after having seen the per-pixel accuracy of 95%.

An upper bound score was calculated on the same U-net's output, and achieved an F1-score of around 0.95. We were quite happy with this result, as it shows the U-net could possibly work very well. We will discuss how this upper bound may eventually be achieved in Section 8.

This upper bound was compared against the analyst, and we found the performance similar. A comparison based on random groupings of donors (Figure 32) showed no significant difference between the two methods or between the sets of donors, as expected. By comparing the analyst and upper bound based on number of donors, we did see a difference. The U-net's performance increased with the number of donors while the analyst's performance was constant (Figure 29). This was an unexpected result, and we have no way to explain this at the moment. More research is needed to conclude the reason behind this definitively.

The most interesting comparison was perhaps the one based on mixture type (Figure 31). While there were some more differences between the analyst and upper bound, they did show the same pattern. As the mixture types got more complex, the performance decreased. The analyst had both better scores for the simpler mixtures, and worse scores for the most complex mixtures. Especially this last observation, that the U-net's upper bound outperforms the analyst in specific cases, shows promise for the method.

8 Discussion

In this chapter, we will discuss all decisions made in this project, and especially the parts that could be improved. We will treat all decisions grouped together by subject: input and deep learning. After that, we will summarise our findings and give recommendations for future research on this subject.

8.1 Input

8.1.1 Adding more labels

On the per-pixel labels, the U-net achieved an accuracy of 95%. This accuracy was slightly higher than Duncan Taylor and David Powers [1] achieved in their first paper (93% on unseen data). This is not a fair comparison, since they also used 5 labels to our 2 labels, and we used a different data set. We cannot predict what the accuracy would be of the U-net in the 5-label case or on their data set. Taylor et al. [2] did increase the number of labels from 5 to 6 in their following paper, among some other changes, to improve the performance of their neural net. We hope this would mean the performance of our U-net would also improve as the number of labels increases.

To annotate the electropherograms with these extra labels, some kind of tool could be developed to make it easier for the DNA analyst to select an area and give it the correct label. It should show the currently known allele labels (both allele names and all pixels selected for that allele), so the analyst can focus on the new information. After this, all remaining pixels will be set to background.

To optimise the annotation process even further, a simple neural network could be applied. We would ask an analyst to annotate a small subset of the electropherograms with the new labels. Then try to make the network predict stutter and pull-up artefacts in the remaining electropherograms decently. Then, ask an analyst to take another look at the discrepancies between the network's prediction and their own annotated labels. They can decide if the prediction was indeed wrong or if the label needs updating. This way we are iteratively improving the labels as well as the results of the neural network.

8.1.2 Cropping electropherograms

The decision to crop the sides of the electropherograms was necessary, because the electropherograms did not have the same length. However, we did make a slightly arbitrary choice to leave only basepairs 50 to 530. Many values could have been chosen for the same purpose. We wanted to choose a starting point after all effects of the primer dimer had disappeared. The stopping point should be anywhere after the rightmost locus, but before the shortest of the electropherograms end. The values of 50 to 530 were also chosen, because this leaves 480 basepairs, or 4800 pixels in each dye, which is easily divisible. Divisibility is useful for the pooling layers in a U-net, in our case we wanted to divide by 2 four times, making 4800 a logical choice.

8.1.3 Normalisation

Applying some form of normalisation turned out to be a good choice in this specific case. It both increased speed and performance of the U-net. We decided to normalise each electropherogram as a whole, as opposed to normalising the entire set of electropherograms based on the overall maximum. We did this, because there can be variation in quality, resulting in lower peaks throughout an entire electropherogram. Then again, the information that a profile is low quality could be useful for the U-net, so it could be looked into more.

Another option was to normalise each dye on its own, because there is a lot of variation in peak heights between dyes. It could be easier for the neural net to recognise peaks if they are all (maximally) the same height. A downside of this method is that pull-up artefacts are related to the height of a peak in a different dye. The relative height of the pull-up with respect to its corresponding peak would be distorted by this normalisation.

Another addition to the normalisation algorithm could be to first cut off all *very high* values and set them to the same maximum. Since peaks above a high enough rfu value are almost definitely allelic, it would remove unnecessary information. The interesting decisions are made close to the baseline, around low values of rfu. After cutting off high values, we could still opt for either normalising or not normalising the data. A problem with this method, is that when peaks are close together, the fluorescent feedback does not always drop to zero, but can show only a dip between two high peaks. If we were to cut off the entire top, we could be merging peaks together, removing the only way to tell them apart.

An important factor to consider is that absolute peak height can be an important tool. Peak height can be used as a factor to judge whether a low peak is an artefact for example. Since it is known with which probability a stutter occurs, Genemarker™ filters out stutters if they have the expected relative height to the allelic peak. This is a property which a neural network might also pick up on (if we did not remove this information).

All in all, each normalisation method has its advantages and disadvantages. Even the option of not normalising has a disadvantage, mainly that rfu values are large numbers and slow down training. We feel confident about our decision to normalise each electropherogram as a whole. Although we have no definitive proof that all other discussed normalisations were outperformed by our choice, we have not seen any signs that the U-net's results were negatively impacted by the normalisation. We did compare the chosen normalisation against no normalisation, and saw an improvement in performance and speed.

8.1.4 Ladder

As we have discussed in Sections 2.3.1 and 5.1, there was some difficulty translating the alleles to the correct peak in the electropherogram and vice versa. With full information, it was doable to select the correct peak from a known allele. However, when identifying alleles purely from the U-net's output, it proved too difficult and we calculated an upper bound. There is a slight mismatch between the allele's bins as given by the Genemarker™ panel file and the resulting peak in the electropherogram. This is solved by Genemarker™ by using a ladder for each run (cf Section 1.2.3).

There are three options to solve this problem. First, a future researcher may think of a better method or algorithm to select the correct alleles based on the U-net's output. If this method is not found, a solution is to either find out how to incorporate the ladder into the (raw) data ourselves, or use a less raw data type from Genemarker™ which has already been analysed and rescaled using the ladders. We believe it is going to be necessary to incorporate the ladders to call all of the alleles correctly.

8.1.5 Different datatype

As discussed in Section 2.2, there were three datatypes available for each electropherogram: our option, an even rawer datatype, and the profile analysed by Genemarker™. With the problems we encountered using the middle option, either using the Genemarker™ profile or a hybrid option could be considered. It is preferable to work with a more raw data type, so the research does not depend (too much) on Genemarker™ specifically and could be applied on electropherograms from other genetic analysis software packages as well. On top of that, not all of the analysis Genemarker™ performs may be needed to solve our problems in allele calling. In the previous section, we mentioned the option of using only the ladders to create a hybrid datatype for our electropherograms.

Other possible useful information that could be added are STR repeat lengths or allele bins. Because these STR lengths vary, the stutter for some STRs will occur a basepair earlier or later than for another STR. This could be confusing our neural network if it happened to pick up that most stutters occur four basepairs before their corresponding allelic peak. To avoid this problem, we could rescale each locus' axis to have the same unit of distance to the expected stutter. Feeding the neural network the precise location of each possible allele (allele bins), removes a lot of area to make decisions on. The network would know in advance that no allelic peaks can exist between loci for example. An added advantage is that it may improve the allele calling on the output of the network as well.

8.2 Deep learning

8.2.1 Parameters of U-net

Most settings of the U-net were very standard choices, e.g. the ReLU activation function ($\max(input, 0)$). The He normal initialiser[21] could not have impacted our results in a big way, because it is only an initialisation. Even if the initialisation was bad, the network would eventually move past it. The number of convolutional and pooling layers (steps) was exactly like the example U-net in Section 4.1. We actually started out with different networks, each with a different number of steps, before we settled on three steps. A larger network needed much more time to train, and a smaller network resulted in too simple predictions (making many mistakes).

Some of the more interesting choices are the parameters of the layers themselves. While it is standard to set the convolution kernel to 3×3, we decided on 3×6 to include all dyes. The 3×3 shape is commonly chosen for two-dimensional images, but our input is actually six interdependent one-dimensional images. On the other hand, we decreased a dimension of the pooling kernel. It is common to half each dimension on a pooling layer (stride of 2 and kernel of 2×2). We chose a stride of 2, but a kernel of 2×1 to avoid pooling in the dye's direction. These choices were made based on intuition and on the data at hand, and could definitely be looked into for improvements.

The most influential change to our U-net, was the average pooling layer. We explained our decision to exchange the max pooling for average pooling layers in Section 6.2. While we believe this was a good decision given our set-up, it may not work as well if the dataset became smaller or larger, or the datatype became different. We have discussed some issues that arose because of the raw datatype we chose for the electropherograms. It is not unthinkable that further research will select a different datatype option to work with. It would be good to reconsider the average pooling layers in that case. We only researched the effect on this specific data with this specific U-net, and cannot predict how it will behave in different cases.

8.2.2 Different neural network

We based our decision for the type of deep learning method on a literature review. We also consulted experts from the FBDA team at the NFI before making our final decision. While we decided on the U-net, another structure that appeared frequently in comparable research were CRNNs. A U-net has convolutional layers, so it is a CNN, but the recurrent layers are still another option to explore. The main goal we would have for these recurrent layers, is to enhance stutter prediction. Recurrent layers are good for finding sequence/time-related features. Although a U-net does solve part of the problem by analysing a full image at once, recurrent layers may prove useful, especially after adding stutter annotations to our labels.

8.2.3 Improving training

Since we were able to achieve good results with little training, we did not make many changes to optimise the training process. We found many good suggestions during the literature review, which could be useful perhaps in future work, e.g. progressive resizing [16]. Taylor et al. [2] improved upon their first proof-of-concept paper by pretraining on specific types of data (e.g. a certain dye). Pretraining certain layers could also be used to speed up training, or by using better hardware or training on an external cluster of computers. Training (and results) could also be enhanced by using a learning rate scheduler. This starts off with a larger learning rate to quickly arrive at a decent solution, and then decreases the learning rate to find the most optimal solution more precisely. We simply used a standard learning rate of 0.001 for training.

8.2.4 Creating more data

An important factor in deep learning, is the amount of available data. We had access to electropherograms of 120 different compositions, of which most were replicated two or three times. In total we could use 350 distinct electropherograms (although the replicates are quite similar). This might sound like a large number, but is not that much in deep learning. Taylor and Powers [1] solved this problem by training their network

on a window of 1206 pixels, on each of the 6000 pixels in one electropherogram. We simply used an entire electropherogram as a single input. It is not easy to create more data, since these require a lot of work from trained specialists. We cannot create more electropherograms on our own, or find more images online due to the specific combination of kit and settings used.

There are some ways to avoid actually creating new data, but rather use existing data to generate similar data. In the specific case of electropherograms, it is important that the dyes are interconnected. We could vertically (keeping all 6 dyes together) cut electropherograms into chunks, and put these chunks together to create a fictitious electropherogram. We would have to be careful with stutters in the horizontal direction, although this probably does not have much effect on a large scale. Or we could copy single peaks or artefacts from one electropherogram into another. This demands some caution in the vertical direction, so we do not copy pull-up artefacts without a corresponding peak for example.

Since we do not know exactly what a neural network learns, it is important to keep the data as realistic as possible. The previously discussed options each had their own downside which would make the generated electropherograms suboptimal. Gordon and Williams [12] used an ingenious technique to add simulated types of noise to heartbeats. Since heartbeat measurement machines have different types of background noise, they added a different type of noise to existing data. This does not remove any of the authenticity from the data, yet can be used to create much more data to train on. It might be interesting for future research to investigate whether this process could be applied to electropherograms.

8.3 Recommendations

We have indicated that our main problem is the allele-calling algorithm on the U-net's output. If one were to build on our research, our advice would be to solve this problem first. It could be interesting to look at new options for the input data type using the ladders. Hopefully, it will not be necessary to use the fully analysed profiles from Genemarker™ . It would be ideal if future research could either enhance the allele-calling algorithm, or design a hybrid datatype somewhere between the more raw and more analysed datatypes.

Another addition to the input data could be to add more labels (stutter and pull-up), which was shown effective in the case of Taylor et al. [1], [2]. This does require a lot of (manual) work from trained analysts, so we would advise to use some sort of preliminary prediction and a handy annotation tool (as discussed in Section 8.1.1) to lessen the workload. We believe the boundaries for cropping are not very interesting to vary, and the normalisation we selected was a good choice, so we would suggest researching other areas first.

The U-net was selected as deep learning method after careful consideration, and performed very well. However, based on the performed literature study, CRNNs did also seem promising. Since we did not implement a CRNN, we cannot be sure that the U-net was the best choice, only that it was a good choice. Future research may look into the differences in performance between these two networks when applied to electropherograms. The U-net we designed was specifically tuned to our data and goal. If something were changed about the datatype or labels, some parameters and/or the structure may need to be tuned again.

We did not have a very large data set, so we did not need to optimise the training process. Since deep learning usually benefits from larger amounts of data, it would be interesting to see how much our U-net would benefit from it. We described some ideas on how to create more (fictitious) data in Section 8.2.4. We believe this could be one of the most impactful improvements to make on this research.

References

[1] D. Taylor and D. Powers, "Teaching artificial intelligence to read electropherograms," *Forensic Science International: Genetics*, vol. 25, pp. 10–18, 2016.

[2] D. Taylor, A. Harrison, and D. Powers, "An artificial neural network system to identify alleles in reference electropherograms," *Forensic Science International: Genetics*, vol. 30, pp. 114–126, 2017.

[3] D. Taylor, M. Kitselaar, and D. Powers, "The generalisability of artificial neural networks used to classify electrophoretic data produced under different conditions," *Forensic Science International: Genetics*, vol. 38, pp. 181–184, 2018.

[4] A. J. Meulenbroek, *De Essenties van forensisch biologisch onderzoek: Humane biologische sporen en DNA*. Uitgeverij Paris, 2009.

[5] M. van Belkom, "NFI," 2020. GitHub repository with code developed for thesis.

[6] C. C. Benschop, A. Nijveld, F. E. Duijs, and T. Sijen, "An assessment of the performance of the probabilistic genotyping software EuroForMix: Trends in likelihood ratios and analysis of type I & II errors," *Forensic Science International: Genetics*, vol. 42, pp. 31–38, 2019.

[7] IBM Cloud Education, "What is deep learning?." https://www.ibm.com/cloud/learn/deep-learning, May 2020.

[8] IBM Cloud Education, "What are convolutional neural networks?." https://www.ibm.com/cloud/learn/convolutional-neural-networks, October 2020.

[9] O. G. Mohammed, K. T. Assaleh, G. A. Husseini, A. F. Majdalawieh, and S. R. Woodward, "DNA base-calling using polynomial classifiers," in *Proceedings of the International Joint Conference on Neural Networks*, pp. 1–5, 2010.

[10] O. G. Mohammed, K. T. Assaleh, G. A. Husseini, A. F. Majdalawieh, and S. R. Woodward, "DNA base-calling using artificial neural networks," in *1st Middle East Conference on Biomedical Engineering, MECBME 2011*, pp. 96–99, 2011.

[11] O. G. Mohammed, K. T. Assaleh, G. A. Husseini, A. F. Majdalawieh, and S. R. Woodward, "Novel algorithms for accurate DNA base-calling," *Biomedical Science and Engineering*, vol. 6, pp. 165–174, 2013.

[12] G. Jimenez-Perez, A. Alcaine, and O. Camara, "Delineation of the electrocardiogram with a mixed-quality-annotations dataset using convolutional neural networks," *Scientific Reports*, vol. 11, p. 863, 2021.

[13] M. Gordon and C. Williams, "PVC detection using a convolutional autoencoder and random forest classifier," in *Pacific Symposium on Biocomputing*, pp. 42–53, 2019.

[14] M. A. Hasan, E. J. Munia, S. K. Pritom, M. H. Setu, M. T. Ali, and S. C. Fahim, "Cardiac arrhythmia detection in an ECG beat signal using 1D convolution neural network," in *2020 IEEE Region 10 Symposium (TENSYMP)*, pp. 352–357, 2020.

[15] J. Xie, X. Aubert, X. Longa, J. van Dijk, B. Arsenali, P. Fonseca, and S. Overeem, "Audio-based snore detection using deep neural networks," *Computer Methods and Programs in Biomedicine*, vol. 200, p. 105917, 2021.

[16] F. Colangelo, F. Battisti, and A. Neri, "Progressive training of convolutional neural networks for acoustic events classification," in *European Signal Processing Conference 2020*, pp. 26–30, 2020.

[17] L. Han and M. R. Kamdar, "MRI to MGMT: predicting methylation status in glioblastoma patients using convolutional recurrent neural networks," in *Pacific Symposium on Biomputing*, pp. 331–342, 2018.

[18] Wikipedia, the free encyclopedia, "U-net." https://en.wikipedia.org/wiki/U-Net, July 2021.

[19] ArcGIS Developers, "How U-net works?." https://developers.arcgis.com/python/guide/how-unet-works/, 2021.

[20] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Fleet D., Pajdla T., Schiele B., Tuytelaars T. (eds) Computer Vision – ECCV 2014. Lecture Notes in Computer Science*, vol. 8689, pp. 818–833, Springer, 2014.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015.

[22] T. Wood, "F-score definition." https://deepai.org/machine-learning-glossary-and-terms/f-score/.

[23] G. Sairam, R. M.K., S. Nithya, and G. Thomas, "An SVM-based algorithm for the prediction and classification of enzymes involved in antibiotic biosynthetic pathways in plant growth promoting pseudomonas species," *Indian Journal of Agricultural Sciences*, vol. 83, p. 75, 2013.

A Figures

The following Figure A.1 shows the difference between peaks and bins for each allele in a single electropherogram.

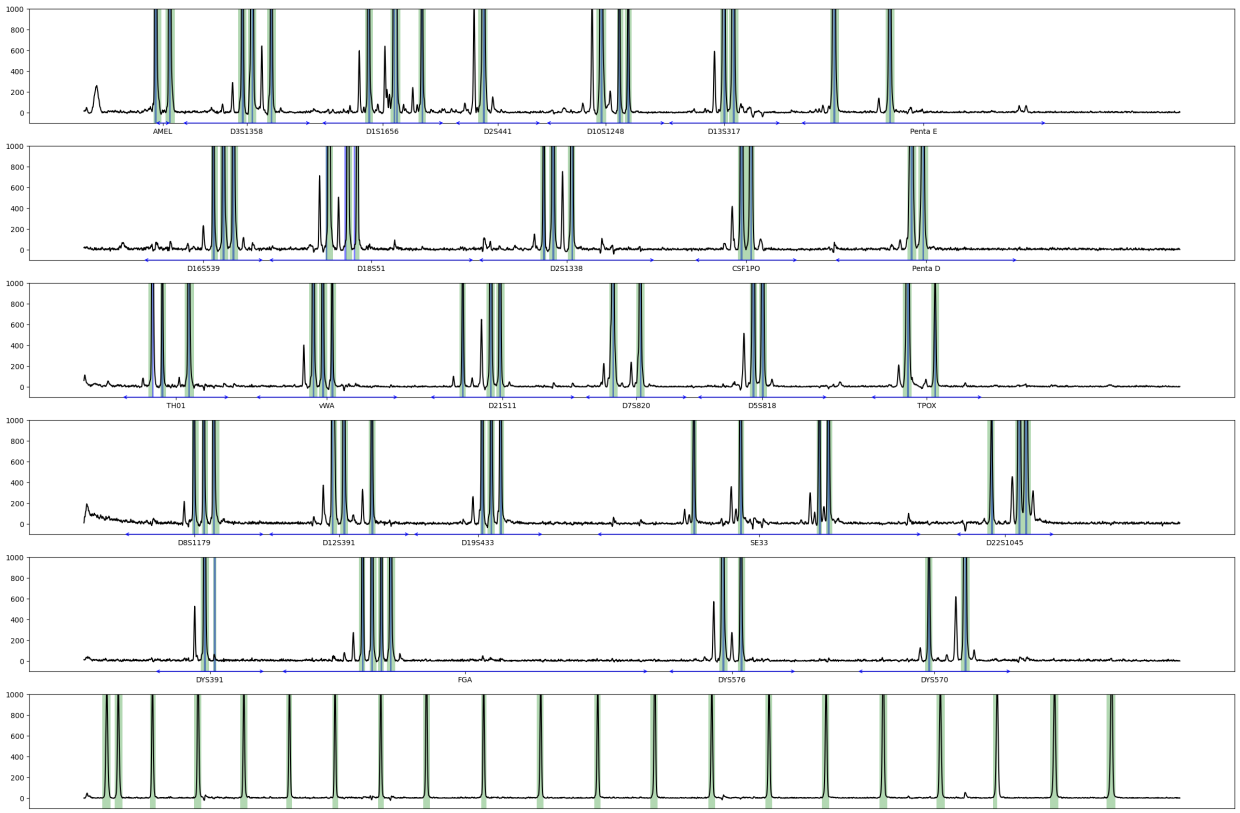


Figure A.1: Bins (blue) and annotated peaks (green) for one full profile. Arrows and text on x-axes are loci and loci ranges respectively. Black line is electropherogram, left axis shows values in rfu.

B Internship report

Internship report will be included exactly as the original on the next few pages, with the exception of page numbers having been added that match the rest of the report.

Exploring and preparing DNA data files for a neural network.

Myrte van Belkom (4448936)

December 2020

1 The NFI

This internship was carried out at the NFI, the Netherlands Forensics Institute. The NFI is a government institute whose main goal is to provide forensic analysis for the police and public prosecutor's office. Another important goal is to increase and share forensic knowledge, in order to increase forensic insight. Examples of this are offering internships, training more forensic scientists, and giving talks about their research. The knowledge at the NFI is always increasing as they are constantly looking for improvements.

2 Problem description

For my internship, the subject lies at the intersection of two fields: FBDA (Forensic Big Data Analysis) and BiS (Biological Traces). The problem was presented to me as both an internship and master thesis subject: Exploring neural networks to enhance DNA analysis. My thesis is going to be to design, create and apply neural networks on the data I have prepared during the internship.

To analyse a DNA sample, the NFI makes use of so-called electropherograms. These are analysed by hand, after which a likelihood ratio calculation determines the weight of the evidence to be reported to the judges. A problem in this process, is that all analysis has to be done by hand. This is very costly and time-consuming. A simple automated process has already been created at the NFI. However, neural networks have been shown to be effective in many problems, and have recently been applied to DNA analyses [4], [2], [3]. To streamline this process, my mentor was wondering whether neural networks could be trained to analyse these electropherograms instead of the analyst.

My internship focuses on how to obtain data in a suitable format for training neural networks from raw DNA measurements (in Python) to prepare them for the neural network. To be able to explain in more detail about my internship, first a small introduction (or refresher) of DNA analysis is needed, which I will give in Section 3.

3 DNA analysis

I will try to keep this DNA introduction as short as possible. Not all of the information is relevant to the project work itself, but it is needed for interpretation of the results. Learning all this new information took place during the first week or two of my internship. Even though I did follow biology in high school, this information was way more specific and in-depth than what I learned. And after 5 years of not using biology, even the general concepts I should have known, had become somewhat vague.

DNA can be found in every cell of your body (except red blood cells). Each cell nucleus has the same 23 chromosome pairs. A chromosome is built from a double DNA string, where the two sides of a DNA string are eachother's opposites. If one side has AGTC, the other sides connects with TCAG. These A(denine), C(ytosine), G(uanine), T(hymine) are called nucleotides, and the A is always opposite to the T, and the C always opposite to the G. One pair of two opposing nucleotides is also called a base pair. On chromosomes, there are certain fixed positions, loci, where the DNA can differ in known ways. The variations on a specific locus are called alleles. Some of these alleles consist of repeated short sequences of nucleotides: Short Tandem Repeats (STR). The different alleles for these loci vary from eachother by the number of repetitions. Since these STRs are easily identified and show plenty of variation over the population, they are ideal to use for DNA analysis.

When trying to identify the DNA in a biological trace, the forensic scientist follows a couple of steps. First, the DNA is extracted as purely as possible. Then only certain identifiable parts (STR's) of the DNA are multiplied many times using PCR (Polymerase Chain Reaction) and five different colour fluorophores are added to specific STR's. A fluorophore is a chemical compound which emits light after excitation (fluorescence). A sixth fluorophore is added to the size standard, which I will explain later on in this section, during the next process: electrophoresis. This sample is led through a very thin tube and hit with a laser. The fluorescent feedback is read by a sensor, resulting in 6 different graphs; one for each colour. This fluorescent feedback is locally very high for the copied STR's and shows up as a peak in the data. So, at this point we have 6 measurements of relative fluorescent units (rfu) against time.

So, the next step is to rescale the measured data's horizontal axis (time) into nucleotides. The longer a string of DNA (the more nucleotides it has), the slower it passes through the tube, and the longer it takes before the DNA passes by the laser. Using a so-called *size standard*, a known DNA sample with its own fluorescent dye, which is run through the machine at the same time, this rescaling is possible. The peaks visible in the size standard are very easily distinguishable from the baseline and have fragments of known sizes. So the time at which these peaks show up, can be linked to a certain length in fragment size. This relation between the time and nucleotide axes can then be applied to all other dyes as well.

Then to identify all peaks in the unknown sample, we use a *ladder*, a mixed sample which (theoretically) has exactly the same amount or DNA of all possibly occurring alleles. All steps in this and the last paragraph are usually automatically performed by Genemarker: a genetic analysis software. The resulting graph of all 6 colours with horizontal axis of nucleotides is called an STR electropherogram. An STR electropherogram shows rfu (on the vertical axis) against fragment size in nucleotides (on the horizontal axis). An example of how a complete electropherogram looks, can be found in Figure 2. Because these images contain a lot of information, they have to be printed quite big to be legible.

When the set of 6 graphs has been loaded into Genemarker, a forensic analyst identifies which peaks he deems to be allelic, and which are artefactual. Small errors in the DNA-replication process can lead to artefacts not related to the actual DNA. One common example of this is stutter: the DNA string folds a little and the copied string skips one repeat (Figure 1). And because the 6 colours' spectra overlap partly, some peaks can appear in more than one colour, although only belonging in one (pull-up). There is a lot of noise at the almost zero level of rfu. Genemarker can filter most of the errors out of the graphs, but does so according to static thresholds.

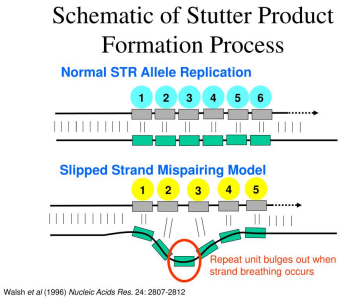


Figure 1: source: https://www.slideserve.com/ivo/310-data-collection-software

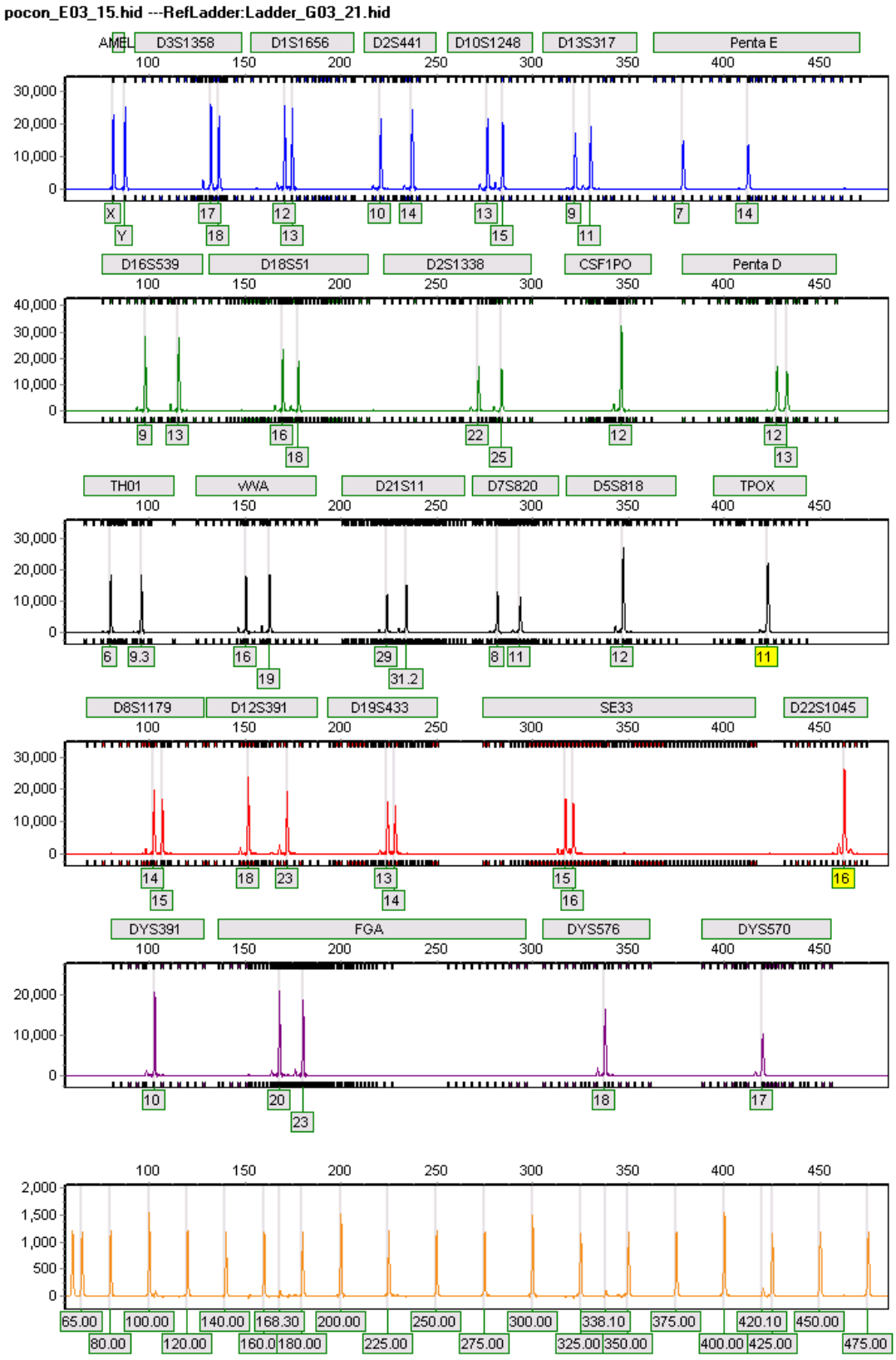


Figure 2: Example of an electropherogram in Genemarker

4 Project

As briefly mentioned in the problem description, Section 2, my mentor was wondering whether a neural network could be trained to identify peaks in an electropherogram; a task currently being performed by hand by trained forensic analysts. This idea originated from a series of papers published by Duncan Taylor and David Powers.

The first paper [4] of this series investigated whether it was at all possible to train an artificial neural network on this type of data. They consider five different labels for each point of the electropherogram: baseline, allele, (backward) stutter, forward stutter, and pull-up. To classify each point, also 100 points to the left and 100 points to the right of it in all six dyes are fed to the neural network. After training on one electropherogram, the neural net already had an accuracy of 93% on an unseen profile (and 98% on the profile it had been trained on).

The second paper [2] tries to improve on the artificial neural network by using a more complex structure and more specific training. They add two layers into the neural net and train the neural net on specific loci or fluorescent dyes. They also add a sixth category (half stutter) and compare the resulting data to genetic analysis software.

The third and last paper they wrote [3] investigates the versatility of their trained neural networks. Under what circumstances you should train another neural net, and when the neural nets can be applied to different types of data.

Duncan Taylor and David Powers mainly aimed to investigate the applicability of neural networks to electropherograms. They use two variations of artificial neural networks, and keep most other choices the same through the papers. There is probably still a lot room for improvement by investigating each of those choices. A different type of neural network may be better suited for this type of data. Convolutional neural networks for example, since they are designed to work well on images, and an electropherogram can be seen as a very simple type of image.

But most of this is of later concern, as those research questions are reserved for my thesis. Before we could train a neural network on data, first the data had to be prepared during my internship. The first step of the thesis project is going to be reproducing the results from Duncan Taylor and David Powers, so I would need to prepare the data in the same format.

4.1 Data files

The DNA data that was made available to me, was not case data, but rather an experimental dataset. The DNA samples had been mixed from known donor DNA in certain ratios. Some mixtures were created from donors who shared many alleles (high allele sharing). This means you would expect it to be difficult to identify the amount of distinct donors, because some alleles will be the same for multiple donors, and only show up once. Other mixtures were low allele sharing, and the rest had randomly selected donors. The available data files were:

- .hid Raw data from genetic analyser
- .txt Trace data of DNA samples (raw or sized)
- .csv Analysts' identified peaks and peak heights
- .csv Donors' DNA profiles
- Word document describing the composition of the DNA mixtures
- .xml Genemarker panel info

If we were able to read out the .hid files, without using Genemarker, that would be ideal. However, this data is extremely raw measurement data, and the files are are not understandable to the human eye. So we use Genemarker to read the .hid file and then immediately export the data without applying any advanced filters or analyses, besides the resizing of the axes. There are two export options that don't require any advanced analysis from Genemarker: raw trace data and sized trace data. My first problem was to figure out the difference between the two, the results of which can be found in Section 5.1.

Then the alleles present in the data could be obtained in two ways. The first option is to use the analysts' identified peaks, which were very nicely ordered in simple .csv files per sample. Second option is to calculate the

theoretical expected peaks from the known sample composition. I had access to .csv files of each donor's DNA profile, as well as a Word document with explanation on the composition of each mixture. I will explain in more detail how the samples were composed and show some examples of the identified and expected peaks in Sections 5.2 and 5.3.

The last file I listed, is the Genemarker panel .xml file. This one was used for a number of things. It contains a huge amount of information on the locations of alleles, which alleles belong to which locus, which locus is measured with which fluorescent dye, and much more details I will not be using. I used this file mainly for plotting purposes, but it is also needed to relate allele names in the peak files to fragment size (positions on the horizontal axis).

The peak data, theoretical or identified, only specifies which alleles are present in an electropherogram, but not the size or shape of this peak. The analysts' file does contain the height of a peak, but not the width. Luckily the Genemarker panel contains the location of allele bins in terms of nucleotides. This gives us at least a general idea of where the peak is, but the bins are still wider than the peaks themselves. As described in Section 4, each point is supposed to get its own label, so I would be needing some kind of peak detection algorithm to identify the single points constituting a peak. Furthermore, Duncan Taylor and David Powers used five different labels in their papers, but these files (after translation of allele to horizontal position and peak detection) only contained two: peak or no peak. So I do not have access to the same labels they do.

5 Results

I will first go into the differences between Genemarker's trace data export options in Subsection 5.1. After that, in Subsections 5.2 and 5.3, I will explain more about how the samples were mixed and how I read out the labels.

5.1 Raw and Sized Trace Data

The first question was, what the difference is between the two export options in Genemarker: raw trace data and sized trace data. These files are exported as tab delimited text files, where each column is one measurement of one dye. The first row contains the sample names per six columns (each sample is measured in six dyes). The second row contains the dye names per column (FL, JOE, TMR, CXR, TOM and WEN). I decided to read out the data in a matrix format, so an $n \times 6$ matrix per sample. What stood out immediately to me, is that the length of the sized data is much shorter than the raw for each sample (about 6000 data points compared to 9000). However, the amount of data points was also not constant. So, to investigate, I plotted both types of trace data in the same figures for each of the 6 colours in a sample. In Figure 3, one such image is shown. Every subfigure contains both raw and sized trace data for the same dye and the same sample.

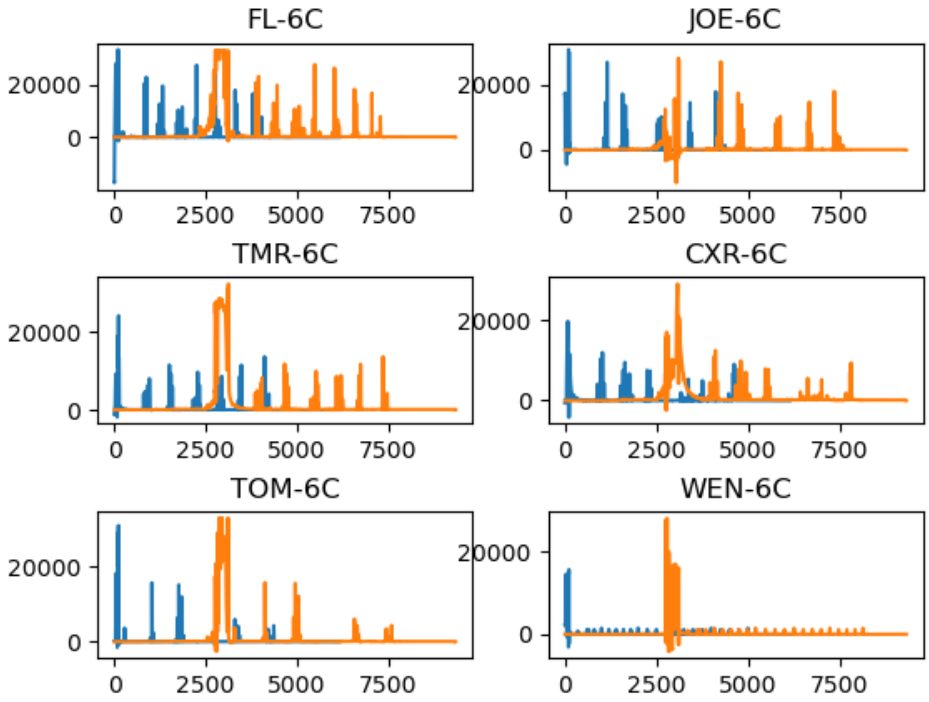


Figure 3: All six dyes of the raw (orange) and sized (blue) trace data.

These dyes all show approximately the same graph in raw (orange) and sized (blue) trace data, only it appears to be translated. An effect you can see very clearly in the WEN dye, the big peak (*primer dimer*) starts immediately for the sized data, and only at about the 3000th data point for the raw data. *Primers* are small single-stranded DNA that act as starting points for PCR. *Primer dimer* is the fluorescent feedback of all the primers that were in the DNA sample for replication during PCR, but weren't used. These primers are all very short and light structures, and easily distinguishable from the actual peaks. Because of their small weight, they show up as the very first peak, since they reach the laser first. Each measurement has such a primer dimer blob as its first peak.

After noticing the translation, I decided to cut off the same (first) part of the raw trace data to be able to see the subtler differences. The next figure, Figure 4, shows only a single dye (JOE) where the raw data was resized to be the same length array as the sized data.

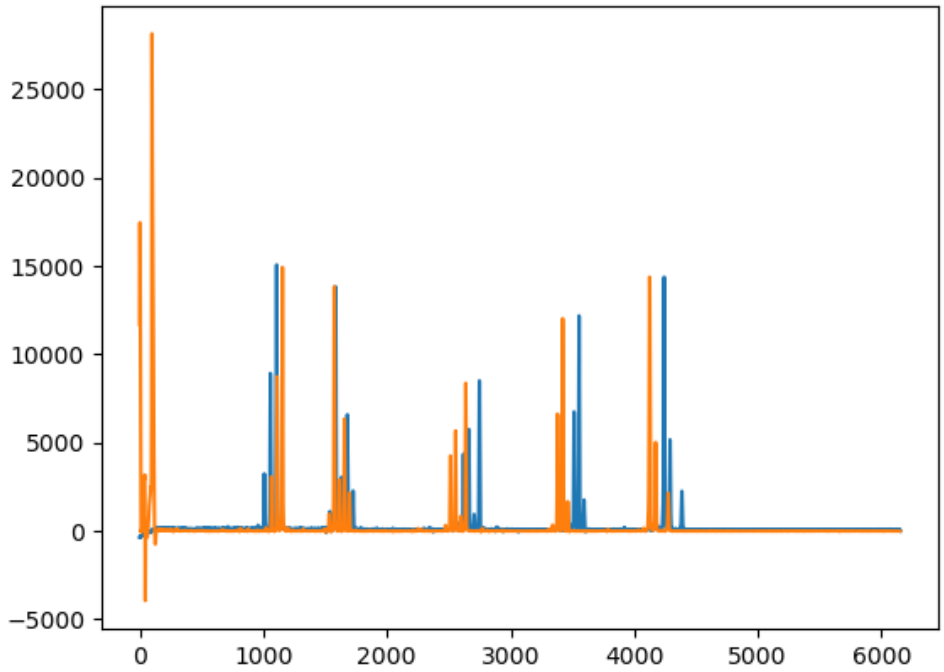


Figure 4: One dye of resized raw (blue) versus sized (orange) trace data, with first part cut off of raw data.

You can see clearly that the graphs are very similar. They have the same amount of peaks and the peaks are almost the same heights. The primer dimer has been completely cut off in the resized raw data (blue), but is still visible in the sized data (orange). Something that stands out, is that they seem to have a different x-axis. For the first few peaks, the raw peaks are before the sized peaks. But later on, the raw peaks *catch up* with the sized peaks, and appear after the sized peaks. To further look into this phenomenon, and what type of rescaling it is, we look at another dye.

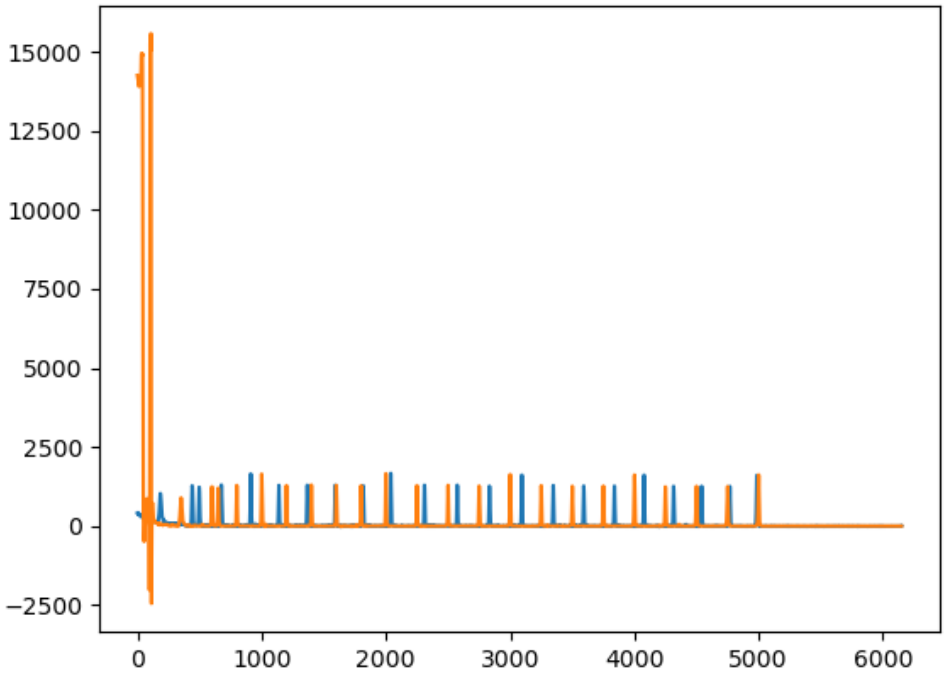


Figure 5: Size standard dye of resized raw (blue) versus sized (orange) trace data.

Figure 5 shows the dye WEN, this dye is reserved for the size standard of a sample. This means that we know where the peaks in this dye are supposed to show up in terms of fragment length. Somewhat surprising is that the raw data peaks (blue) first appear before, then on, then after, and then again on the sized data peaks (orange). This means that it is not a linear transformation from raw to sized data or vice versa.

If you look closely at the peaks of the sized trace data, you can see a peak precisely at 1000, 2000, 3000, 4000 and 5000. I verified this, using a simple peak detection algorithm, and the peaks indeed show up exactly at those data points. These peaks at every thousand data points, are supposed to show up at every 100 nucleotides in the size standard, and a few steps in between (every 20 nucleotides up to 200 and every 25 nucleotides from 200 on). From this, I concluded that the sized trace data had been resized to a fragment length axis using the size standard. So the sized trace data has 10 measurements per nucleotide. This discovery made plotting the samples in an electropherogram a lot simpler, since I could now link a data point to a value on the fragment length axis. Exporting sized or raw trace data from Genemarker takes the same amount of time, so I decided to use the sized data from here on.

5.2 Reading out identified peaks

Now that I knew how to identify the data from the trace data, I wanted to compare it to the peaks that should be visible in the figure. Each of the samples has already been read by an analyst, and he had denoted the alleles and heights of each peak. Below, figure 6 shows the peaks (*) the analyst has identified in the sample for one dye (FL).

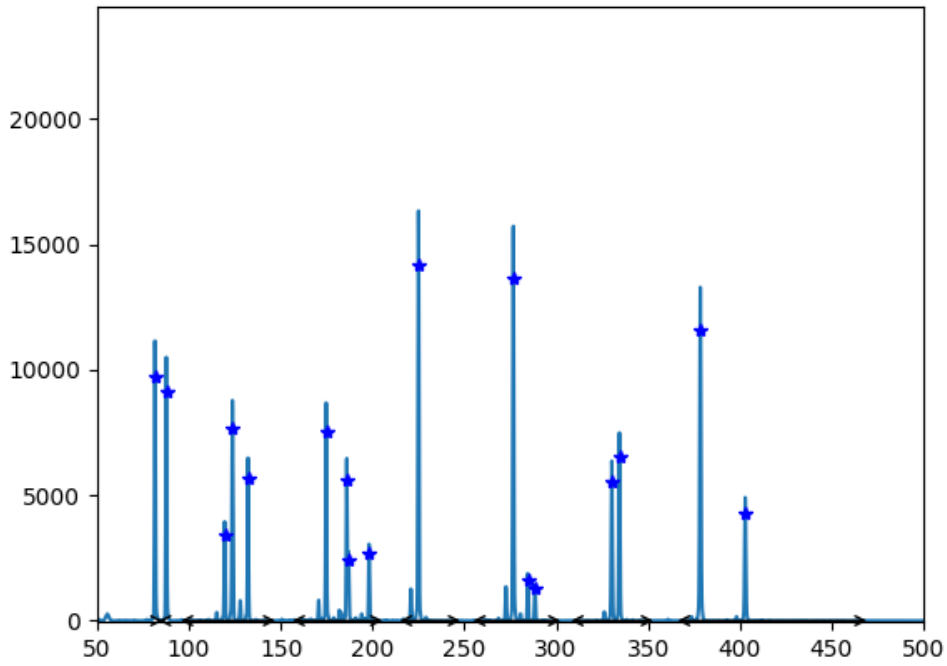


Figure 6: Analyst's identified peaks (*) and measured DNA data for one dye.

The heights are also plotted exactly as the analyst has written them down. The difference in height is most likely due to the filters Genemarker uses. Although there is a small difference in height of the peaks that were measured, and the peaks that the analyst identified, they do clearly describe the same mixture. The lower peaks are much closer to what the analyst described than the higher peaks.

On the horizontal axis you can see the marker ranges plotted. These show the range where alleles for the same locus should appear. This information was taken from the Genemarker panel information. If this was a single person profile, we would expect 1-2 peaks per marker range (one for each chromosome in a pair). One peak in a marker means that this person has the same allele on both chromosomes. This also explains why the single peaks are so much higher, because those have twice the amount of genetic material per allele.

5.3 Reading out expected peaks

All of the experimental data I was given, was composed from known donors in known proportions. These samples are coded by a short sequence denoting the amount of different donors, the ratios of DNA between donors and which set of donors the sample came from. For example, the next figure shows a mixture of two donors where the main donor to secondary donor ratio is 2:1.

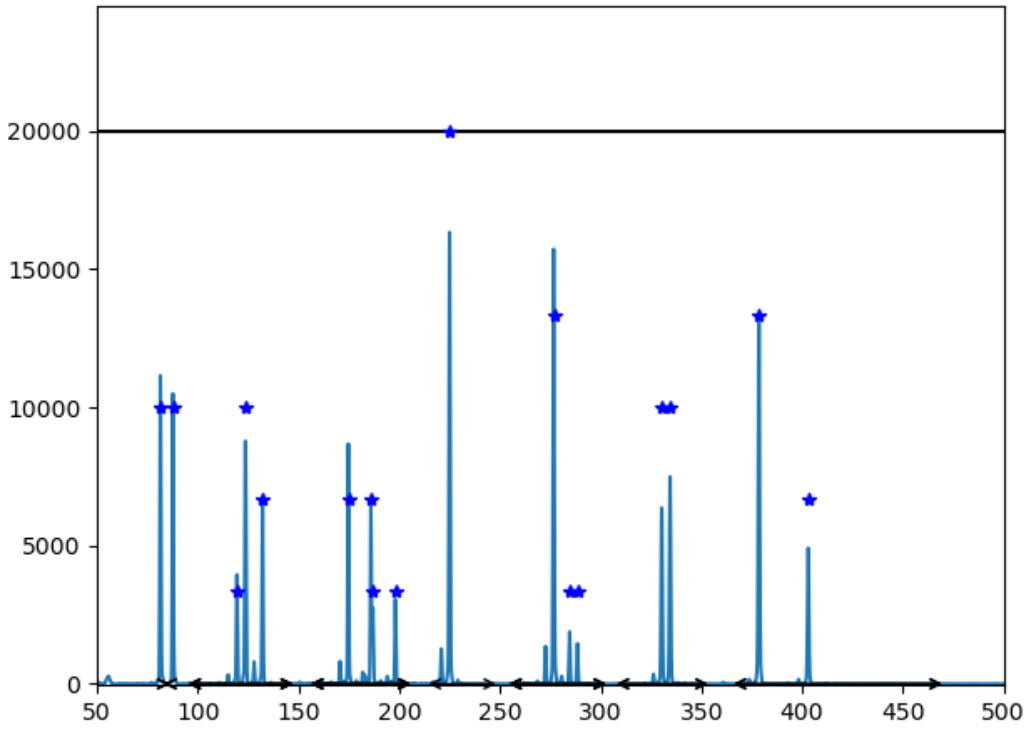


Figure 7: Expected peaks (*) and measured DNA data for one dye.

In Figure 7, I tried to visualize the expected peak heights. There is no precise relation between the amount of genetic material and the height of a peak. Obviously, the more material, the higher the peak. However, a certain amount of DNA does not relate to a certain height in rfu and can also differ between measurements. DNA also becomes more susceptible to decay the further it shows on the horizontal axis.

This specific sample was made up of two donors, one of which was added twice as much DNA as the other one. I added together the relative contributions to the total weight of the DNA sample, to come up with relative heights of the peaks. The horizontal black line at 20000 rfu shows the maximal possible peak (*) height. This means that the * on this black line represents an allele that each donor on each chromosome. The large peak to the right of it, around the 275th nucleotide, is almost the same height in the measured data, yet the amount of DNA is not even 75% of the DNA for the first peak.

It was very good to see that the expected peaks matched the measured data so well horizontally. This means that the data was read out properly, and the Genemarker panel file matches the data. The panel contains the allele and marker ranges for all alleles that can be measured using this set-up. Figure 8 shows all allele and locus bins that Genemarker has stored in its panel. At zero level, the locus bins are between the squares for all 5 dyes. The size standard obviously does not contain alleles, so was not plotted in this case. At the height of one, all allele bins are plotted. These bins are incredibly small, they have a width of about one (nucleotide), but it varies a little between alleles.

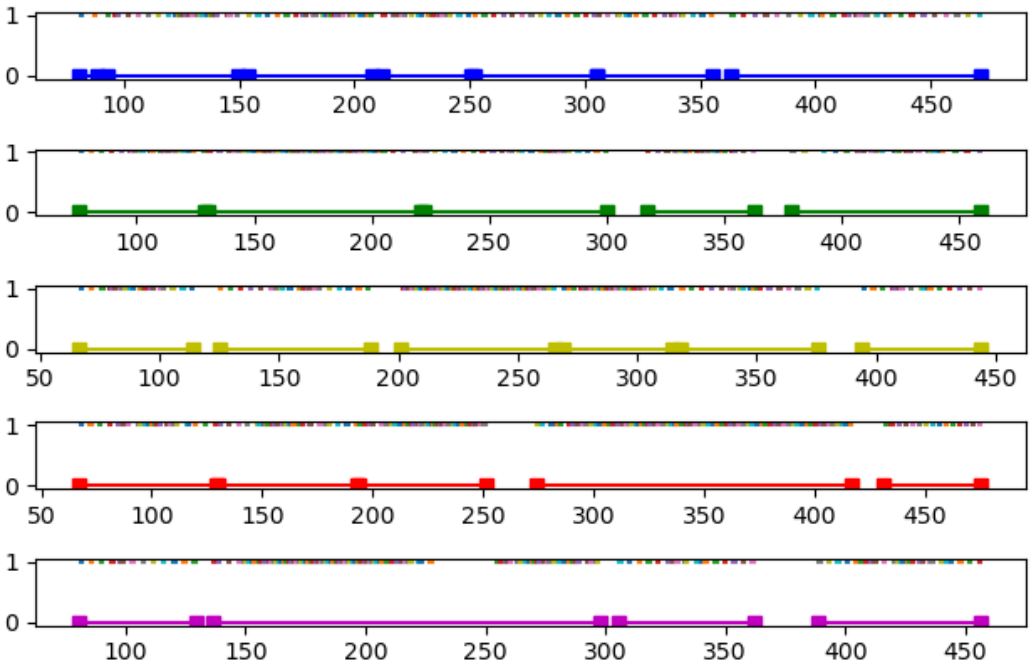


Figure 8: Allele and locus bins

5.4 What I have learned

I have learned a lot these past months, for example knowledge of DNA, better programming skills and machine learning basics. My mentor advised me to use Github for version control of my code, and I'm very glad he did. Version control software is not something we learn to use in the bachelor Applied Mathematics, even though it would have been very useful. I think there are some more advanced programming courses in the master, mainly aimed at the CSE specialisation. Unfortunately, I did not have any room left in my exam programme to add these courses, but luckily my internship did provide an opportunity for more programming.

During the only required programming course in Applied Mathematics, we learn to use Python with its built-in IDLE (Integrated Development and Learning Environment) [1]. At this internship, I learned how to use Python in Pycharm instead, and virtual environments using Anaconda. Never having known about other IDE's (Integrated Development Environments), I didn't know what I was missing until I tried it. Being able to rename a variable everywhere in your code, even distributed over multiple files, with a single click is ideal. Anaconda gives you an overview of the packages you have installed for each virtual environment, and makes it easier to install or uninstall packages.

More specific skills I developed are learning to deal with lots of different types of data file. I also got some experience in presenting my project and interim results to different audiences. The digital meeting environments and screen-sharing sometimes provided difficulties, but never too much. Doing this internship right before my thesis, gave me the room to figure out how to work independently. This was especially challenging because of the Corona virus measures, and having to do the largest part of the work at home.

Lastly, I would like to add what I have learned during the internship besides the project work itself. I was included into the team FBDA from the start, which not only made me feel very welcome, but also gave me a taste of the day-to-day working environment in FBDA. They adapt an agile work structure, which means that they have short meetings to start each day and see how everyone is doing. Every week they share some larger updates on their projects, and every few weeks they present their final results. It was interesting to see what actual researchers were working on.

References

[1] *Python's IDLE*. URL: https://docs.python.org/3/library/idle.html.

[2] Duncan Taylor, Ash Harrison, and David Powers. "An artificial neural network system to identify alleles in reference electropherograms". In: *Elsevier Forensic Science International: Genetics* 30 (2017), pp. 114–126. ISSN: 1872-4973. DOI: http://dx.doi.org/10.1016/j.fsigen.2017.07.002.

[3] Duncan Taylor, Michael Kitselaar, and David Powers. "The generalisability of artificial neural networks used to classify electrophoretic data produced under different conditions". In: *Elsevier Forensic Science International: Genetics* 38 (2018), pp. 181–184. ISSN: 1872-4973. DOI: http://dx.doi.org/10.1016/j.fsigen.2018.10.019.

[4] Duncan Taylor and David Powers. "Teaching artificial intelligence to read electropherograms". In: *Elsevier Forensic Science International: Genetics* 25 (2016), pp. 10–18. ISSN: 1872-4973. DOI: http://dx.doi.org/10.1016/j.fsigen.2016.07.013.

Appendix

On the following pages the developed code is included, the appendices A: classes.py, B: reading_functions.py, C: plotting_functions.py and D: main.py.

A classes.py

This file contains all data classes and some global variables.

```
from dataclasses import dataclass
from typing import List, Dict
import numpy as np


@dataclass
class Dye:
    """ Class for fluorescent dyes of genetic analyzer."""
    name: str           # example: 'FL-6C'
    plot_color: str     # example: 'b'
    plot_index: int     # index of which of 6 subplots when all dyes\
                        # are plotted in the same image


# Currently between classes so it can be used within classes
BLUE = Dye('FL-6C', 'b', 1)
GREEN = Dye('JOE-6C', 'g', 2)
YELLOW = Dye('TMR-6C', 'y', 3)
RED = Dye('CXR-6C', 'r', 4)
PURPLE = Dye('TOM-6C', 'm', 5)
LADDER = Dye('WEN-6C', 'k', 6)


@dataclass
class Allele:
    """Class for each allele that can be identified."""
    name: str       # example: 'X' or '17'
    mid: float      # horizontal position, example: '87.32'
    left: float     # left side of bin from mid (0.4 or 0.5)
    right: float    # right side of bin from mid (0.4 or 0.5)


@dataclass
class Locus:
    """Class for locus, stores Alleles per locus in dict."""
    alleles: Dict[str, Allele]      # example of entry: '18': Allele()
    name: str                       # example: 'AMEL'
    dye: Dye                        # dye that locus is on
    lower: float                    # lower boundary of marker
    upper: float                    # upper boundary of marker


@dataclass
class Peak:
    """Class for an identified or expected allele peak.
    Has everything needed for plotting."""
    name: str       # Using "locus_allele" because it makes dict access easy
    x: float        # horizontal location of peak
```

```
    height: float    # height of peak
    dye: Dye         # dye of peak


@dataclass
class Sample:
    """
    Class for samples, data is (nx6) matrix of all 6 colours
    """
    name: str        # example: '1A2'
    data: List
    color_list = [BLUE, GREEN, YELLOW, RED, PURPLE, LADDER]


@dataclass
class Person:
    """ Class to store alleles a Person has. """
    name: str             # name is A - Z, letter used to identify person
    alleles: List[str]    # list of 'locus_allele' names


@dataclass
class PersonMixture:
    name: str                        # for example: "1A2"
    persons: List[Person]            # list of Persons present in mix
    fractions: Dict[str, float]      # fractional contribution of each person in mixture
    def create_peaks(self, locus_dict):
        """Returns list of peaks expected in mixture and their relative heights"""
        peak_list = []
        peak_dict = {}
        # add X and Y by hand (all samples are male)
        X_and_Y = locus_dict['AMEL'].alleles
        X = X_and_Y['X']
        Y = X_and_Y['Y']
        peak_list.append(Peak("AMEL_X", X.mid, 0.5, BLUE))
        peak_list.append(Peak("AMEL_Y", Y.mid, 0.5, BLUE))
        # iterate through persons in mix
        for person in self.persons:
            # iterate over their alleles
            for locus_allele in person.alleles:
                try:
                    peak_dict[locus_allele] += self.fractions[person.name]
                except:
                    peak_dict[locus_allele] = self.fractions[person.name]
        # now we have a dictionary of the height of the alleles
        # all that's left is to store corresponding peaks in list
        for locus_allele in peak_dict:
            locus_name, allele_name = locus_allele.split("_")
            locus = locus_dict[locus_name]
            allele = locus.alleles[allele_name]
            x = allele.mid                          # store x_location
            height = peak_dict[locus_allele]        # store rel. height
            dye = locus.dye                         # store dye
            new_peak = Peak(locus_allele, x, height, dye)
            peak_list.append(new_peak)               # append peak to list
        return peak_list
```

```
@dataclass
class AnalystMixture:
    """ Class to store peaks identified in mixture. """
    name: str               # name of mixture, '1A2' for example
    replicate: str          # where 1 is donor set, 2 is #donors, A is mixture type and 3 is
    peaks: List[Peak]       # list of peaks


# Global variables
PICOGRAMS = np.array([[300, 150, 150, 150, 150],
                      [300, 30,  30,  30,  30],
                      [150, 150, 60,  60,  60],
                      [150, 30,  60,  30,  30],
                      [600, 30,  60,  30,  30]])

TOTAL_PICOGRAMS = np.array([[450, 600, 750, 900],
                            [330, 360, 390, 420],
                            [300, 360, 420, 480],
                            [180, 240, 270, 300],
                            [630, 690, 720, 750]])
```

B reading_functions.py

This file contains all functions used for reading out data files.

```
import pandas as pd
import xml.etree.ElementTree as eT
from src.classes import *


def txt_read_data(filename: str):
    """ Function to read data files\
    Returns a list of sample names, colors, \
    and the data itself as matrix."""
    textfile = open(filename, "r")  # open text file
    texts = textfile.read()         # read entire content
    texts = texts.split("\n")       # split into lines
    # lines 1 and 2 are not interesting
    titles = texts[2].split('\t')                       # get titles of files
    titles = [item for item in titles if item != '']    # remove empty entries after splittin
    colors = texts[3].split('\t')                       # only needed for width of lines
    data = np.zeros((len(texts[4:]), len(colors)))
    counter = 0                                         # counter is needed for line number
    for elt in texts[4:]:
        new = np.array(elt.split('\t'))     # split into words
        new[new == ''] = 0                  # if empty string, make zero
        data[counter, :] = new              # store into data array
        counter += 1
    # now pour contents into separate sample dataclasses
    sample_list = []
    for i in range(len(titles)):
        new_sample = Sample(titles[i].split('_')[0], data[:, 6*i:6*i+6])
        sample_list.append(new_sample)
    return sample_list
```

```
def xml_read_bins():
    """Read xml file for bins of each allele, \
    returns dictionary of information"""
    tree_file = eT.parse("data/PPF6C_SPOOR.xml")
    root = tree_file.getroot()
    locus_dict = {}
    # root[5] is the node with loci, rest is panel info
    for locus in root[5]:
        locus_name = locus.find('MarkerTitle').text
        # to translate the numbers in xml file to dyes
        temp_dict = {1: BLUE, 2: GREEN, 3: YELLOW, 4: RED, 5: LADDER, 6: PURPLE}
        dye = int(locus.find('DyeIndex').text)
        lower = float(locus.find('LowerBoundary').text)
        upper = float(locus.find('UpperBoundary').text)
        # store info so far in Locus dataclass
        new_locus = Locus({}, locus_name, temp_dict[dye], lower, upper)
        # add all alleles to locus
        for allele in locus.findall('Allele'):
            allele_name = allele.get('Label')
            mid = float(allele.get('Size'))
            left = float(allele.get('Left_Binning'))
            right = float(allele.get('Right_Binning'))
            # store in Allele dataclass
            new_allele = Allele(allele_name, mid, left, right)
            # add to alleles dict of locus
            new_locus.alleles[allele_name] = new_allele
        # add created locus to locus dict
        locus_dict[locus_name] = new_locus
    return locus_dict


def csv_read_persons(donor_set):
    """reads all profiles from given donor set (1,2,3,4,5 or 6)"""
    filename = 'data/donor_profiles/Refs_dataset' + str(donor_set) + '.csv'
    donor_peaks = pd.read_csv(filename, dtype=str, delimiter=";")
    person_list = []                              # initialize lists
    alleles = []
    person_name = donor_peaks['SampleName'][0]    # get first donor name
    for index, row in donor_peaks.iterrows():     # iterate over all alleles
        if row[0] != person_name:                 # we have arrived at a new person
            # store up to now in Person dataclass, start new list
            person_list.append(Person(person_name, alleles))
            alleles = []
        person_name = row[0]                      # first entry is person name
        locus = row[1]                            # second entry is locus name
        allele1 = locus + "_" + row[2]            # third entry is first allele
        allele2 = locus + "_" + row[3]            # fourth entry is second allele
        alleles.append(allele1)
        alleles.append(allele2)
    return person_list


def person_contributions(person_list, number_of_donors: int, mixture_type: str):
    """Calculates relative contributions of each person based on mixture type"""
```

```
    # Temporary dict to translate letter to row
    letter_to_number = {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4}
    mixture_row = letter_to_number[mixture_type]     # type of mixture determines the row
    person_dict = {}                                 # initialize list and dict
    persons = []
    parts = PICOGRAMS[mixture_row]                   # global variables for contributions
    total = TOTAL_PICOGRAMS[mixture_row, number_of_donors - 2]
    for i in range(number_of_donors):
        frac = parts[i]/total/2                      # divide by 2 because 2 alleles per locus
        person_dict[person_list[i].name] = frac      # add fraction to person
        persons.append(person_list[i])
    return person_dict, persons


def make_person_mixture(mixture_name, locus_dict):
    """Uses person_contributions and csv_read_persons to create expected peaks in person mixt
    donor_set, mixture_type, donor_amount = mixture_name        # can be "1A2" for example
    donor_amount = int(donor_amount)
    person_list = csv_read_persons(donor_set)
    person_fracs, persons = person_contributions(person_list, donor_amount, mixture_type)
    person_mix = PersonMixture(mixture_name, persons, person_fracs)
    peaks = person_mix.create_peaks(locus_dict)
    return peaks


def csv_read_analyst(sample_name, locus_dict):
    """Read csv file of analyst's identified alleles returns list of corresponding peaks"""
    results = pd.read_csv("data/analysts_data_filtered/"+str(sample_name)+"_New.csv")
    name = results['Sample_Name'][0]     # to start iteration
    sample_name, replicate = name.split('.')
    mixture_list = []                    # initialize big lists
    peak_list = []                       # initialize small lists
    for index, row in results.iterrows():
        # iterate over all rows, because each row contains the peaks for one allele
        if name != row[0]:                           # then start new sample
            sample_name, replicate = name.split('.')
            mixture_list.append(AnalystMixture(sample_name, replicate, peak_list))
            peak_list = []                           # empty list
        name = row[0]                                # then set name to current sample name
        for i in range(2, 12):
            # go over the 10 possible locations of peak identification
            if str(row[i]) == row[i]:
                # append value only if non-empty
                # empty entries are converted to (float-type) NaN's by pandas
                # so str(row[i]) == row[i] filters out empty entries
                locus = locus_dict[row[1]]
                allele = locus.alleles[row[i]]
                x_value = allele.mid     # location on x_axis
                height = row[i+10]       # heights are 10 indices further than
                dye = locus.dye          # their corresponding allele names
                new_peak = Peak(locus.name+"_"+allele.name, x_value, height, dye)
                peak_list.append(new_peak)
    mixture_list.append(AnalystMixture(name, replicate, peak_list))
    return mixture_list
```

C plotting_functions.py

This file contains all functions used for plotting the read out data files.

```
import matplotlib.pyplot as plt
from scipy.signal import find_peaks
from src.classes import *


def plot_sample_markers_6C(sample: Sample, locus_dict: dict):
    """Plots sample and markers in 6C plot"""
    plt.figure()
    # iterate through all loci to plot markers
    for key_locus in locus_dict:
        locus = locus_dict[key_locus]                # get locus class object
        plt.subplot(6, 1, locus.dye.plot_index)      # plot in correct color location
        # plot bar at level 0 with squares as endpoints to show marker
        # might want to change style of endpoints
        plt.annotate(s='', xy=(locus.lower, 0), xytext=(locus.upper, 0), arrowprops=dict(arro
        # plt.plot([locus.lower, locus.upper], [0, 0], color = locus.dye.plot_color, marker =
    for i in range(6):
        plt.subplot(6, 1, i + 1)
        current = sample.data[:, i]
        plt.plot(np.linspace(0, len(current)/10, len(current)), current, str(sample.color_lis
        plt.xlim([50, 500])
    plt.suptitle(sample.name)
    plt.tight_layout()
    plt.subplots_adjust(top=0.9)
    plt.show()


def plot_analyst(peaks: list, sample: Sample, locus_dict):
    """uses both the analysts identified peaks and sized data \
    for comparison to plot both in one image for each color"""
    for j in range(6):
        plt.figure()
        plt.title(str('filename: '+sample.name+', dye: ' + str(sample.color_list[j].name)))
        plt.xlim([50, 500])      # to cut off primer dimer
        current_plot = sample.data[:, j]
        # plot measured data
        plt.plot(np.linspace(0, len(current_plot)/10, len(current_plot)), current_plot)
        # to scale y-axis somewhat close to data
        plt.ylim([-50, max(current_plot[1000:])*1.5])
        # iterate through all alleles in mixture
        for peak in peaks:
            dye = peak.dye
            if dye.plot_index == j+1:
                locus = locus_dict[peak.name.split("_")[0]]
                plt.annotate(s='', xy=(locus.lower, 0), xytext=(locus.upper, 0), arrowprops=d
                plt.plot([peak.x], [peak.height], str(dye.plot_color + "*"))  # add colour
        plt.show()


def plot_analyst_6C(peaks: list, sample: Sample, locus_dict):
    """uses both the analysts identified peaks and sized data \
    for comparison to plot both in one image for each color"""
    plt.figure()
```

```
    for j in range(6):
        plt.subplot(6, 1, j + 1)
        plt.xlim([50, 500])     # to cut off primer dimer
        current_plot = sample.data[:, j]
        # plot measured data
        plt.plot(np.linspace(0, len(current_plot)/10, len(current_plot)), current_plot, sampl
        # to scale y-axis somewhat close to data
        plt.ylim([-50, max(current_plot[1000:])*1.5])
        # iterate through all alleles in mixture
    for peak in peaks:
        dye = peak.dye
        plt.subplot(6, 1, dye.plot_index)
        # get marker bins
        locus = locus_dict[peak.name.split("_")[0]]
        # plot marker bins
        plt.annotate(s='', xy=(locus.lower, 0), xytext=(locus.upper, 0), arrowprops=dict(arro
        # plot peaks
        plt.plot([peak.x], [peak.height], "k*")  # add black colour
    plt.suptitle(sample.name)       # set title
    plt.tight_layout()              # ensures subplots don't overlap
    plt.subplots_adjust(top=0.9)    # ensures title doesn't overlap plots
    plt.show()


def plot_expected(peaks: list, sample: Sample, locus_dict: dict):
    """uses both the theoretical actual relative peaks and \
    sized data for comparison to plot both in one image"""

    for j in range(6):
        # makes one separate figure per dye
        plt.figure()
        plt.title(str('filename: '+sample.name+', dye: ' + sample.color_list[j].name))
        current_plot = sample.data[:, j]
        # cut off primer dimer
        plt.xlim([50, 500])
        # amount to multiply relative peak height with
        max_rel = max(current_plot[1000:]) * 1.5
        # set y_max at 1.5 times max peak
        plt.ylim([-50, max_rel])
        # plot max height relative points
        plt.hlines(max_rel, 0, 500)
        # plot measured data
        plt.plot(np.linspace(0, len(current_plot)/10, len(current_plot)), current_plot)
        # iterate through dict to plot all peaks as *
        for peak in peaks:
            dye = peak.dye
            if dye.plot_index == j + 1:
                color = dye.plot_color
                locus = locus_dict[peak.name.split("_")[0]]
                plt.annotate(s='', xy=(locus.lower, 0), xytext=(locus.upper, 0), arrowprops=d
                plt.plot([peak.x], [peak.height * max_rel], str(color + "*"))  # add colour
        plt.show()


def plot_expected_6C(peaks: list, sample: Sample, locus_dict):
    """uses both the analysts identified peaks and sized data \
```

```
    for comparison to plot both in one image for each color"""
    plt.figure()
    for j in range(6):
        plt.subplot(6, 1, j + 1)
        plt.xlim([50, 500])     # to cut off primer dimer
        current_plot = sample.data[:, j]
        # amount to multiply relative peak height with
        max_rel = max(current_plot[1000:]) * 1.5
        # set y_max at 1.5 times max peak
        plt.ylim([-50, max_rel])
        # plot max height relative points
        plt.hlines(max_rel, 0, 500)
        # plot measured data
        plt.plot(np.linspace(0, len(current_plot)/10, len(current_plot)), current_plot, sampl
    # iterate through all peaks in mixture
    for peak in peaks:
        dye = peak.dye
        plt.subplot(6, 1, dye.plot_index)
        # get marker bins
        locus = locus_dict[peak.name.split("_")[0]]
        # plot marker bins
        plt.annotate(s='', xy=(locus.lower, 0), xytext=(locus.upper, 0), arrowprops=dict(arro
        # plot peaks
        plt.plot([peak.x], [peak.height * max_rel], "k*")  # add black colour
    plt.suptitle(sample.name)       # set title
    plt.tight_layout()              # ensures subplots don't overlap
    plt.subplots_adjust(top=0.9)    # ensures title doesn't overlap plots
    plt.show()


## UNDERNEATH ARE MOSTLY UNUSED ###
def plot_data(sample: Sample):
    """Simple plot of all colors of one sample in the same figure"""
    plt.figure()
    for i in range(6):
        plt.plot(sample.data[:, i], label=str(sample.color_list[i]))
    plt.legend()
    plt.title(sample.name)
    plt.show()
    return None


def plot_6C(sample: Sample):
    """Plots one combined plot of all 6 colors of one sample"""
    plt.figure()
    plt.suptitle(sample.name)
    for i in range(6):
        plt.subplot(6, 1, i + 1)
        plt.plot(sample.data[:, i])
        plt.title(sample.color_list[i])
    plt.show()
    return None


def plot_sizestd_peaks(sizestd):
    """The goal of this function was to determine the factor needed \
```

```
    for resizing the sized data to base pairs\
    Input is one size standard array, output was a plot"""
    peaks, rest = find_peaks(sizestd, distance=200)
    plt.figure()
    plt.plot(sizestd)
    print(peaks)
    plt.plot(peaks, sizestd[peaks], "*")
    plt.show()
    return peaks


def plot_markers(locus_dict):
    """Just a quick function to test marker boundaries"""
    plt.figure()
    # iterate through all loci
    for key_locus in locus_dict:
        locus = locus_dict[key_locus]            # get locus class object
        plt.subplot(6, 1, locus.dye.plot_index)     # plot in correct color location
        # plot bar at level 0 with squares as endpoints to show marker
        plt.plot([locus.lower, locus.upper], [0, 0], color=locus.dye.plot_color, marker="s")
        # iterate through all alleles
        for key_allele in locus.alleles:
            allele = locus.alleles[key_allele]  # get allele class object
            start = allele.mid - allele.left    # calculate where it starts
            end = allele.mid + allele.right     # calculate where it ends
            plt.plot([start, end], [1, 1])       # plot allele bin at level 1
    plt.show()
```

D main.py

This file contains an example usage of some available functions.

```
from src import data_prep_functions as df, plotting_functions as pf, reading_functions as rf

if __name__ == '__main__':
    # first create a list of all samples
    sample_list_1 = rf.txt_read_data("data/trace_data/TraceData1.txt")
    sample_list_2 = rf.txt_read_data("data/trace_data/TraceData2.txt")
    sample_list = sample_list_1 + sample_list_2
    # now we get all panel information from the Genemarker file
    locus_dict = rf.xml_read_bins()
    # plot some samples with marker bins
    for i in range(10, 13):
        current = sample_list[i]
        current_name = current.name
        pf.plot_sample_markers_6C(current, locus_dict)
        # Everything underneath
        # plot samples with analyst's identified peaks
        replicas = rf.csv_read_analyst(current_name, locus_dict)
        # now have a list of the analyst's identified peaks + heights for all replicates
        # currently always picks first replica
        pf.plot_analyst_6C(replicas[0].peaks, current, locus_dict)
        # now plot with actual peaks
        peaks = rf.make_person_mixture(current_name, locus_dict)
        pf.plot_expected_6C(peaks, current, locus_dict)
```