2020

How Well a Neural Network System Could Interpret the Number of Contributors to a DNA Profile.

(THESIS REPORT)

Submitted By
PARTHO PROTIM GHOSH

Supervisors
Prof. David M W Powers <David.powers@flinders.edu.au>
Dr. Duncan Taylor <Duncan.taylor@flinders.edu.au>

Submission Date
24 June 2020

Submitted to the College of Science and Engineering in partial fulfilment of the requirements for the degree of Master of Science (Computer Science) at Flinders University, Adelaide, Australia.

I, Partho Protim Ghosh, declare that all works in this project is my authentic work unless otherwise stated or referenced.

Signed,

Partho Protim Ghosh

On

24/06/2020

Acknowledgements

Thanks to Prof. David M. W. Powers and Dr. Duncan Taylor, the supervisors of this thesis and all their great support and advise. Specially I would like to thank Duncan and Forensic Science South Australia for providing me population frequency databases and other databases to generate the DNA profiles and for a proper guideline of entire project.

Table of Contents

List of Figures and Tables

Abstract

Machine learning approaches and artificial neural network (ANN) are very popular approaches to improve the efficiency in many fields including economics, biology, finance, image processing and gaming. It reduces the manual effort where complex interpretation and analysis are required. The artificial neural network (ANN) and machine learning (ML) are quite new approach in the field of DNA analysis and there are lots of prospects which can be visualised by this technology. There are several goals included in the research work: Create a system using the extensions of ANN such as Recurrent Neural Networks (RNN), or Convolutional Neural Networks (CNN) that could interpret the number of contributors to a DNA profile. The research can be divided into four components: Simulate realistic DNA profile with a probability in accordance to their population frequency database (provided by FSSA named Australia_Caucasian.csv); Simulate DNA profile for single contributor and mix the single profiles to generate mixture of DNA profile for different contributors (Number of contributors in mixture can be 2/3/4/5); Divide the profiles in 21 smaller parts (Each part represents a Locus and DNA profile consists with 21 locus); Prepare the mix profiles and single contributor profiles to train the Neural Networks; Train the Convolutional Neural Networks (CNNs) with the input profiles (Mix Profiles) and output profiles (Individual Profiles) for all 21 Loci and evaluate the performance.

Keywords: DNA, DNA profiling, Artificial Neural Network (ANN), Convolutional Neural Network (CNN), Forensics, Bioinformatics

1. Introduction:

Machine learning approaches and artificial neural network (ANN) are very popular approaches to improve the efficiency in many fields including economics, biology, finance, image processing and gaming. It reduces the manual effort where complex interpretation and analysis are required. The artificial neural network (ANN) and machine learning (ML) are quite new approach in the field of DNA analysis and there are lots of prospects which can be visualised by this technology. DNA profile of more than one contributor are considered as a mixture of DNA and it is collected from more than one person or contributor. The most complex step in forensics is to interpret the individual contributor from the DNA mixture. It is essential that a number of contributors is assigned to a DNA profile before it can be analysed. This task is currently carried out by experts, using their experience.

Forensics Science South Australia (FSSA) is a government organisation and handles crime case work. FSSA working in four different field, that are pathology, biology, toxicology and chemistry. In the real world of investigation, DNA analysis plays a vital role. A typical case work of FSSA involves few steps. Where police attend the crime site and collected the DNA samples from suspected and from suspect and crime scene. After taking the DNA samples, these are sent to forensic biology research lab to extract the DNA from the samples. Then DNA profile is generated and compared with the other samples to find out the individual who may have contributed DNA to the sample.

In this research, it has two components. 1) Simulate DNA profile and 2) Applied artificial neural network (ANN) approach to improve the interpretation, specifically with respect to how many contributors there might be in the mixture. In the first part of research, mixed DNA profiles are simulated from some DNA database (Australian Caucasian). After that different ANN approach have used to train and investigate the performance the number of contributors that the DNA profile has originated from. Several deep learning approaches including Recurrent Neural Network (RNN), Convolutional Neural Network (CNN) have been tested on generated datasets. And finally, the performance of research evaluated on CNN based implementation. Figure 1 represents an EPG of three contributor mixed DNA profile. (Taylor et al. 2013). The EPG has developed based on the observed height of 16 locus/regions of three known contributors.

Figure 1 : EPG of three contributor mixed DNA profile [1]

2. Literature Review:

2.1 Identifying numbers of contributor in DNA profile:

Majority of DNA profile interpretations have followed few steps. In the first step, DNA profiling has been implemented and after that classifier algorithm impose on the data set. However, no single algorithm is perfectly working on every type of classification. The performance of algorithm depends on the size of data, quality and characteristics (Marciano et al. 2017).

Two computations are frequently used in the field of forensics to evaluate the evidential weight of DNA profile data. These are likelihood ratio (LR) and the combined probability of inclusion (CPI). Recently few more calculations have been adopted by few laboratories. That are Random match probability (RMP). Taylor et al. [1] adopted likelihood ratio (LR) approach in their analysis on interpretation of single source and mixed DNA profile. The mathematical methods have programmed as software and validate the results by handwritten calculations. The proposed method produced an LR. As there are no true LR available that's why it is not possible to examine the result by some true answers. The following practical tests are done in that case. 1) Examination of interpretations of mixtures of known contributors (ground truth). 2) Comparisons against other methods and/or human judgement. There are few limitations happen when a large number of artefacts is allowed through the manual EPG review process. Also, currently the number of contributors must be specified by the user prior to analysis and number must be same in both numerator and denominator.

2.2 Classification of fluorescence in a DNA profiles:

Even before the data from a DNA profile can be used to determine a number of contributors, the raw signal from the laboratory instruments must be processed. There may be some scope to link the processing of raw data and the determination of number of contributors to a DNA profile to achieve the best performance. Taylor et al. [3] proposed a novel solution, where the artificial neural network can be trained on data of different sources (i.e. single sourced profiles or mixed DNA profiles) and from different laboratory conditions. In this research they extended their previous work where they demonstrated an artificial neural network (ANN) that was trained on two good quality reference EPGs to classify the data in the 6-FAM dye lane and then applied to a third good quality EPG with a reasonable success. In the current research, they extended their work by increasing the number of training data, increasing the range of training EPG quality, improving the quality of artificial neural network (ANN), training a series of 10 ANNs for different areas of the EPG and Coupling the predictions of the ANNs with a peak detection algorithm proposed by Woldegebriel et al. [4]. In the research they proposed probabilistic peak detection algorithm based on Bayesian framework for forensic DNA analysis. The proposed method worked with raw electropherogram data from a laser-induced fluorescence multi-CE system. The main finding of the research was that a system of ANNs trained on both single sourced and mixed DNA sample electropherograms performed comparably to ANNs trained on, and applied to, only one data type.

2.3 Using Artificial Neural Network:

Taylor et al. [5] discussed about the demonstration of the use of artificial neural network (ANN) which can be trained to read electropherograms and show that it can generalise to unseen profiles. The research was done based on five categories including Baseline, Allele, Stutter, Pull-up and Forward Stutter. The result of that investigation was very remarkable. On the training dataset the ANN was able to learn to correctly classify approximately 98% of the 12,000 scans. When the model was then applied to the test dataset the performance was slightly lower at approximately 93%, but still generally high.

Lawless C. [6] discussed in his article about dispute that arisen controversy in the DNA profiling technology that known as ‘low-template DNA’ method. Here author described about the technologies being used by forensics technology. LT-DNA was one of them, this technology was introduced by UK Forensic Science Service (FSS) and quickly it was hailed as a cutting age technology to sort out criminal cases. It represents relatively minor and incremental adaptions to DNA profiling technology. It also deploys a probabilistic Bayesian method that is a popular approach in the field of forensic science. It employed an increased number of cycles of polymerase chain reaction (PCR), it means it copy the samples repeatedly. PCR cycle has been carried out 28 times of in the ordinary DNA profiling. On the other hand, FSS method involved 34 repetition in a profile. Due to low amount of biological material in the initial samples, this repetition has improved the quality of profiling. This study reviled the technological limitations of LT-DNA and its high level of multivalence. A set of distinctions were drawn like basic distinctions between 'valid' science verses efficient 'pathological science' and the 'new' verses the 'same' technology. Also, most important part of the discussion was, they examined LT-DNA as a tool and as a set of conditions to profiling the DNA.

Cowell et al. [7] discussed different artefacts of DNA profiling and the analysis of forensic DNA mixture. In this research, authors presented a statistical model for a quantitative peak information from an EPG model and its significant impact on the criminal case that handled by the forensic team. Here EPG model has generated by the previously used algorithm and imposed some important artefacts on it. The model has some unknown parameters and its estimated maximum likelihood in the presence of multiple unknown contributors in the sample, multiple errors has generated, and it exploit a Bayesian network representation of the model. In their model, they choose potential contributor in a DNA mixer and choose specific maker and allele and this model is described the peak heights. In the model they represented stutter by decomposing the individual contributions to peak heights by adding each contribution. After that they calculated dropout in the mixtures where some alleles are not calculated.

Dakhli et al. [8] presented a new approach for DNA sequence classification. In this research, they proposed a solution based on using Wavelet Neural Network (WNN) and the k-means algorithm. The performance of WNN is depending on the proper demonstration and implementation of WNN structure. The approach uses the Least Trimmed Square (LTS) and the Gradient Algorithm (GA) to solve the architecture of Wavelet Neural Network (WNN). The initialization of WNN was solved by the method Trimmed Square (LTS), which is applied for wavelet candidate by selecting Multi Library of the Wavelet Neural Networks (MLWNN) for constructing the WNN. This LTS method has used to choose best wavelet from the library. On the other hand, the gradient algorithm has used to train the dataset of WNN. In the paper, they classify the system in three phases. The first on is transformation that composed by three sub steps. Binary codification of DNA sequences Fourier Transform and Power Spectrum Signal Processing. The second step is approximation, and it is encouraged by the use of Multi Library Wavelet Neural Networks (MLWNN) and finally classification of DNA sequence has implemented. The proposed solution in paper can be used to classify the DNA sequence of organism into many classes. Also, this solution can be used to extract significant biological knowledge.

In another research Cheng et al. [9] discussed about segmentation of DNA using simple recurrent neural network. In the study they reported the strong correlation between protein coding region and the error prediction during the simple recurrent network to segment genome sequences. In the experiment they used SARS genome to demonstrate the procedure of training and the method of achieving the results. Also, the HA gene of influenza A subtype H1N1 analysed in the similar way. H1N1 subtype has analysed using both supervised and unsupervised simple recurrent network.

Most current approaches to mixture deconvolution require the assumption that the number of contributors are known by the analysts. Determining the number of contributors can be complex when the mixture contains 3 or more contributors. There are many algorithms and approaches focussed on the getting good results from a neural network to interpret the number of contributors to a DNA profile. Marciano et al. [2] focused on a probabilistic approach for determining the number of contributors in a DNA profile that contain more than one person DNA sample. In the research they trained, tested and validate using electrical data obtained from 1405 non-simulated DNA mixture samples comprised of 1-4 contributors and generated from a combination of 20 individuals. In the same research, five candidate machine learning algorithms including k-Nearest Neighbors (k-NN), Classification and Regression Trees (CART), Multinomial Logistic Regression (Logit), Multilayer Perceptron (MLP), Support Vector Machine (SVM) are used to classify the problems. The performance of the algorithms often depends on the size, quality and characteristics of the associated training data. In the research, overall results show over 98% accuracy in identifying the number of contributors in a DNA mixture of up to 4 contributors.

In this paper Zang et al. [10] have used an artificial neural network approach to establish a unified model of DNA analysis to solve classification problem. In this model, they used a parallel logic that is completely different from the traditional computer neural network. In the traditional neural network model, weights between neurons and neurons are eventually stable by continuously adjusting and this process is basically maintaining a serial. On the other hand, in the parallel DNA model, the weights are calculated by determining all the possible weights that are suited to all samples. In the building neural network, the weights are calculated by the step by step modification through the training dataset. DNA computing in this paper, divided the interval into sub intervals. For example, the interval can be divided equally into 20 sub intervals. DNA computing employs a linear pattern classifier and it can be categorised into three modules. 1) Coding DNA modules, 2) Learning process and 3) Classification procedure. In the learning process, for the K-layer, firstly generated the input and pass them to the weights. Secondly, performed the weighted sum ration. After that according to the given output, selected the possible output chains. In the classification process, they produced the input of given samples, pass the input through the weights of the first layer. After that performed the weighted summation reaction and generated the input for the next level. The proposed algorithm has some advantage compared to traditional algorithms. Here the weights are calculated by all possible weights combination instead of calculating neurons to neurons.

2.4 Recurrent Neural Network and LSTM (Long Sort-Term Memory):

Recurrent Neural Network (RNN) is a novel algorithm for subsequentially data and currently used by Apple's Siri and Google voice search [11]. RNN is the algorithm that can remember its input because of the internal memory unit. And that's why it is a novel solution for subsequentially data like financial data, DNA sequence and time series data. Recurrent Neural Network (RNN) was introduced by David Rumelhart in their work on learning representations by back-propagation errors in 1986 [23]. On the other hand, Long Sort-Term Memory (LSTM) was invented by Hochreiter and Schmidhuber in 1997 [22]. Long Sort-Term Memory (LSTM) is a kind of RNN that contains internal memory cell in the layer. And it can store information for long period of time for further processing.

Cheng et al. [12] have used simple recurrent neural network (RNN) for segmentation of DNA profile. In this research, they discovered notable correlation between the coding sequence of protein and error prediction. To find out the genome sequence here RNN was used. A simple recurrent neural network (also called Elman network) that consists with three-layer neural network with an edition "context neuron" in first layer. This network performs with steam state that can handle a subsequent prediction task. The model of the recurrent neural network is like below in Figure 2.

Figure 2: Structure of Recurrent Neural Network with context layer [12]

Without any prior biological knowledge researcher in this project, suggested a new technology to read genome sequence. On the other hand, processing the ATCG sequence, the result was consistence and properly matched with finding from biologists. This new technology could be used in studying of complex genome which are still a mystery to biologists. By analysing the error from this study, it would be easier for researcher to identify which part of genome are worth to study. The result of the method can be used to identify the artificial DNA segments from natural segment.

In biomedical data analysis, DNA sequence classification is a key work in computational framework. And in recent years of studies, several machine learning techniques have been introduced to classify the DNA sequence. In a recent study, Bosco et al. [13] proposed two different deep learning-based architectures that classify the DNA sequence automatically. The proposed methods have evaluated on public data set of DNA sequence for different classification tasks. In this research, they evaluate the performance with CNN, CNN-NT, LSTM, LSTM-NT models over 10 folds cross validation with same data set. Here recurrent neural network consists with 6 different layers. Where first embedding layer followed by a max-polling layer size of 2. where input was one-hot encoding vector. The max-pooling layers reduce the size of vector and reduce the computations of the next layers. After max-pooling layer they implemented a recurrent layer that is working as LSTM. And it is can execute input from left to right and produces an output vector size of 20. Then configured another max-pooling layer and at the end two fully connected layers. The design of RNN model are like below:

Figure 3: Structure of Recurrent Neural Network used in DNA sequence classification [13]

Here same dataset collected from bacteria species applied on CNN and LSTM models. And CNNs performed better in four simple classification tasks. On the hand, LSTM worked better than convolutional neural network (CNN).

Ling et al. [14] proposed a recurrent neural network (RNN) solution for modelling biological solution. Their proposed method is an alternative of currently using ordinary differential equations (ODE) for modelling accurate temporal dynamics of networks. It could be solution over the limitation of ODE method that includes complexity in kinetic parameter estimation and numerical solution with complex equations. The proposed RNN model can estimate the parameter easily from data and efficiently examine the network behaviour. Modelling p53-

Mdm2 using RNN involved few steps in the described research. 1) Understanding the protein connections related in the system. 2) Evaluate the performance of existing solutions. 3) Developing RNN that represent ODE parameters. 4) Generate and validate the data for RNN and 5) Investigate the parameters and performance of proposed model.

Recurrent Neural Network (RNN) is an approach that is appropriate for sub sequential data like weather data, financial data, DNA data and time series data. Chen et al. [15] proposed a Reinforced Recurrent Neural Network model to forecast multi step ahead flood. In the research, there are several parts are involved. 1) Construct a Reinforced Recurrent Neural Network (R-RTRL NN) model to improve future flood predictions. 2) The R-RTRL NN system rapidly update its parameters with latest values to improve the predictions. 3) BPNN and another two neural network model are used here to compare performance with R-RTRL systems. 4) Models are constructed to make multi step ahead (MSA) forecasts for time series and flood series. For comparison purpose three neural network developed here (two dynamic neural networks and one static neural networks). The examination of result showed that numerical and experimental evaluation of R-RTRL systems was more accurate over other developed neural network systems.

Recurrent neural network is different than feedforward neural network. It is not only dealing with initial inputs but also works on internal state space. Internal state space tracks the data what are being processed already. Raza et al. [16] have developed a Recurrent Neural Network (RNN) based gene regulatory system (GRS). It is actually a hybrid version of an existing system named Kalman filter for weight update in backpropagation through time training algorithm. According to the research, RNN is a noble approach that provides a great combination of biological closeness and mathematical flexibility to GRS. Recurrent neural network has a capability to capture complex, non-linier and dynamic relations between variables [16]. The implementation has also compared with the traditional system. And got the better results compared to others. Further, 5% Gaussian noise added on the data to check the effect on result. And the effect was negligible.

2.5 Convolutional Neural Network:

Convolutional Neural Network is most influential extension of neural network. And it is widely using by technology giants in their products. Facebook using on their automatic tagging information, Instagram uses on their search infrastructure [20]. Lecun et al. [21] worked on the modern convolutional neural network (CNN) for the first time by 1990s, that was inspired by the neocognitron. In the research, authors used CNN for handwritten character recognition.

Figure 4: Structure of Convolutional Neural Network [24]

The convolutional layers are the major parts of Convolutional Neural Network (CNN). CNN calculate L- one dimensional convolutions between the kernel vectors w^l , of size 2n+1, and the input signal x: [13]

Another important component of Convolutional Neural Network is max-pooling layer. It is a non-linier layer that partitions the input vectors and classify the non-overlapping values and for each sub region the maximum values considered as output.

Convolutional Neural Network [CNN] based approach is the efficient approach for image classification. And deep learning-based approach is wise than traditional approaches because it works on deep features [17]. Liu et al. [17] proposed a pixel-based attention map using convolutional neural network (CNN). In traditional image recognition process has few major steps including image collection, image classification (i.e. local binary pattern, support vector mechanics, histogram of oriented gradient + SVM), image recognition and feature extraction. In this research, developed P_VggNet solution worked better than the traditional approaches.

Modelling the properties of DNA sequence is an important but a challenging task in the field of biotechnology. According to the research of Quang et al. [18] this task is even more difficult for the class of DNA that are non-coding DNA. A noble approach for the non-coding DNA could be the useful for both basic science and translational research. Because over 98% human genome are non-coded [18]. In this research, authors proposed a noble solution named DanQ, a hybrid convolutional neural network (CNN) and a bidirectional LSTM recurrent neural network model to predict non-coding function from DNA sequence. In the LSTM based recurrent neural network layers capture the dependencies between the motifs to learn regulatory grammar to improve the performance. On the other hand, convolutional layers capture the regulatory motifs. This model contains 320 convolutional kernels for 60 epochs. Also evaluate the average multi-task cross entropy loss for validation after each epoch to monitor the performance of training. Here in this model, dropout to randomly set proportion of neuron has included from the max polling and BLSTM layers to each training step. Authors have compared the performance of developed DanQ model to a LR baseline model and the published Deep-SEA model. According to summery of research, DanQ a powerful method that can predict the function of DNA directly from sequence. Also, the powerful method of this architecture allows to simultaneously learn motifs.

Convolutional neural network is a sort of feed forward ANN where the neurons are connected locally and share the parameter mechanism. Without the definition of parameters, CNN can able to extract features from raw input dataset and keep tracing the number of model parameters by a series of convolutional layers and a series of pooling layers. And in a standard CNN model, number of convolutional layers and pooling layers operated followed by one or more fully connected layers after the last polling layer. In a research, Du et al. [19] proposed a deep learning solution (Convolutional Neural Network) named DeepSS. That consists two modules. DeepSS-C module used to classify the splice sites and DeepSS-M module used to detect the splice site sequence pattern.

3. Importance of the Research:

3.1 Motivation of Research:

The most complex challenge in the field of Forensic DNA analysis, is to separate the DNA of individual contributor from a DNA mixture. DNA analysis are widely using by the forensic analysists to identify the crime sources. According to the literature reviews that have discussed earlier, current developed approaches can work as expected for DNA mixers of 3-4 contributors.

Also, there are lots of manual task are involved in the analysis and detecting of the DNA containing cell on the sample. Sometimes cells needed to count manually by the analysts. And sometimes quality of the samples is not same in a DNA mixer. So, there is a scope to work on this issue. On the other hand, it can be wise to develop an artificial neural network system that can be solution of these limitations. The neural network (NN) is the well-recognised approach to analysis the complex task like analysis DNA sequence. Sometimes it can be difficult and complex to sort out the samples from the DNA mixture of more than one contributor.

A current limitation to the machine learning of number of contributors is that the data is highly complex, and the relative amount of training data available is limited. However, we have models of DNA profile behaviour (which we use in other areas of DNA profile evaluation) that should allow very realistic DNA profiles to be simulated for any number of contributors. So, it will be interesting to work on the trailed system that can able to simulate mixtures of varying complexity / quality / proportion etc as training data set with test / validation sets being real data.

3.2 Goal of Research:

There are several goals included in the research work. Create a system using the extensions of ANN such as Recurrent Neural Networks (RNN), or Convolutional Neural Networks (CNN) that could interpret the number of contributors to a DNA profile. The research can be divided into four components:

- Simulate realistic DNA profile with a probability in accordance to their population frequency database (provided by FSSA named Australia_Caucasian.csv).
- Simulate DNA profile for single contributor and mix the single profiles to generate mixture of DNA profile for different contributors (Number of contributors in mixture can be 2/3/4/5)
- Divide the profiles in 21 smaller parts (Each part represents a Locus and DNA profile consists with 21 locus). Prepare the mix profiles and single contributor profiles to train the Neural Networks.
- Train the Convolutional Neural Networks (CNNs) with the input profiles (Mix Profiles) and output profiles (Individual Profiles) for all 21 Locus and evaluate the performance.

3.3 Scope of Research:

- Simplifying the DNA profiles.
- Examine the classifiers for different number of contributors.
- Increase the performance of Neural Network by training with more DNA profiles.

4. Methodologies:

4.1 Dataset:

The system has been trained, tested and validated with simulated DNA mixture samples. Dataset consist with input and output data. Output data contain the allele numbers of profile and input data contain weight, height, expected back strutter ratio, observed back strutter ratio, expected forward strutter ratio, observed forward strutter ratio and population allele frequency. For both mix DNA profile and individual profile, dataset has divided into 21 parts. And trained 21 Neural networks with it. Each neural network has represented each locus (Total number of locus is 21).

4.2 Machine Learning Algorithms:

Based on the finding on the literature review, no single algorithm is perfectly working on every types of classification. The performance of algorithms depends on the size of data, quality and characteristics (Marciano et al. 2017). To find out the optimal solution I have worked on the following algorithms including Convolutional Neural Network (CNN), Principal Component Analysis (PCA), Non-negative matrix factorization (NMF) and Normalization have applied on training dataset.

4.3 Training the NNs:

Training has been conducted using tools developed by Python (Framework: Tensorflow) and Keras model. Every instance data has divided into training, test and cross-validation. All 21 neural networks have been trained and evaluated with simulated DNA profiles. Dataset generation tool also developed using Python 3.7.

5. Project Achievements:

Project has two major components. Simulate DNA profiles (Both mixed profile and individual profile), applying classifiers on the input and output files to train Neural Networks. And Train Convolutional Neural Network (CNNs) with input and output profiles. And then evaluate the performance.

5.1 Simulation of DNA Profiles:

In the first part of the research, the simulation of DNA profile has been implemented. Numpy and Pandas frameworks of Python 3 have been used here to implement. And below are the steps that are used to implement.

1. The first step is to pick alleles at each locus. This will involve picking alleles with a probability in accordance to their population frequency (here I have used an Australian Caucasian database). Here I have picked two of these alleles, with replacement from the list of alleles that have some frequency > 0. These may end up being the same allele

(in which case it represents a person who is homozygous at that locus), or they may be two different alleles (in which case it represents a person who is heterozygous at that locus). If two different alleles are picked then the convention is to order them in ascending order (e.g. if you picked a 13 and then a 11 then your genotype is [11,13]). At first 21 locus were chosen and stored it into another database 'AustralianCaucasian_21_data' and after that below scripts have applied to pick alleles:

```
for i in range(1, len(header_list)):
    index =
abs(AustralianCaucasian_21_data[header_list[i]] -
0.2).idxmin()
    AustralianCaucasian_21_data[header_list[i]][index] =
AustralianCaucasian_21_data[header_list[0]][index]
    index =
abs(AustralianCaucasian_21_data[header_list[i]] -
0.2).idxmin()
    AustralianCaucasian_21_data[header_list[i]][index] =
AustralianCaucasian_21_data[header_list[0]][index]
    print(header_list[i])
```

Below are the picked alleles for the 21 loci.

Loci	Alleles
CSF1PO	10, 11
D10S1248	15, 16
D13S317	8, 12
D16S539	9, 13
D18S51	12, 14
D19S433	13, 15
D1S1656	15, 16
D21S11	29, 30
D22S1045	11, 16
D2S441	10, 14
D3S1358	16, 17
D5S818	10, 13
D7S820	8, 11
D8S1179	12, 14
DYS391	9, 12
FGA	21, 22

SE33	28.2, 29.2
TH01	6, 7
TPOX	9, 11
Yindel	1, 2.2
vWA	16, 18

Table 1: Picked alleles of 21 Loci

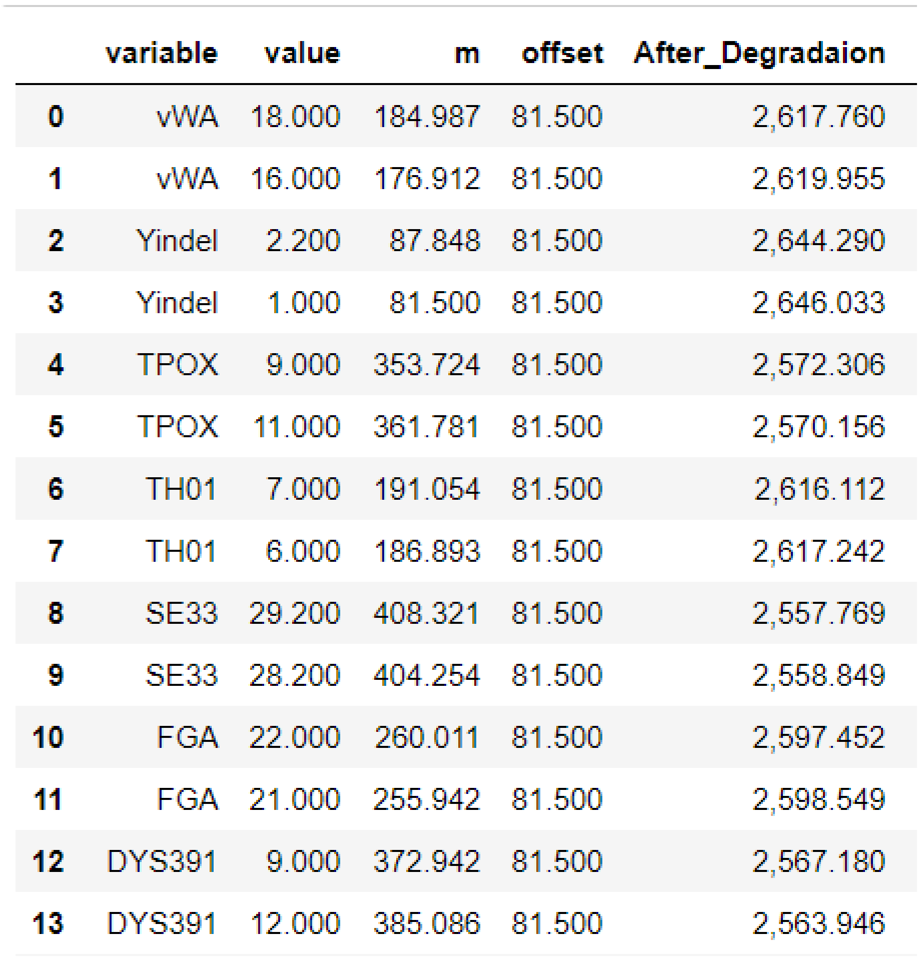
2. At this stage I have two alleles chosen for each of 21 loci. The next step is to pick a DNA amount. This amount is generally pretty free to pick anything, but to me more realistic we want to bias the choice a bit to lower values. I used an exponential (0.0004) between 100 and 10000. In DNA profiles the peak heights are measures in relative fluorescent units (rfu), which hare arbitrary values, but we can consider them as amount of DNA. We have designated DNA amount as T (for Template DNA).
   Having picked DNA amounts I will now be able to give heights to the alleles that picked in step 1. If the locus is heterozygous (2 different alleles were chosen) then they each get the DNA amount that just picked to make 2 peaks (each peak gets a dose, call this X, of 1). If the locus is homozygous (one allele was chosen twice) then that one allele gets twice the DNA amount you just picked, to make 1 peak (dose, X = 2). Using the below numpy's random exponential library the value has generated.

```
T = np.random.exponential(scale=1/0.0004, size=1)
```

3. At this stage molecular weights have assigned to each of the peaks. These are the sizes of the DNA fragments that have been amplified. To do this I have used a database 'GlobalFiler_SizeRegression.csv', which gives the slope and intercept of a line that translates allele designation (i.e. the values have picked in step 1) to molecular weight (which designated as m).
   Next level of degradation for the profile have picked. Choose a value from uniform distribution U[0,0.003]. Let's refer to degradation as D. The amount that degradation affects peak height can be determined by:

```
peak_height_after_degaradtion_applied = peak_height_before_degaradtion_applied *
exp[-D(m-offset)]
```

where offset is the lowest molecular weight peak in the profile. Here the offset value is 81.5. The values of peak height after degradation are as follows for just some of the loci shown in table 1:



	variable	value	m	offset	After_Degradaion
0	vWA	18.000	184.987	81.500	2,617.760
1	vWA	16.000	176.912	81.500	2,619.955
2	Yindel	2.200	87.848	81.500	2,644.290
3	Yindel	1.000	81.500	81.500	2,646.033
4	TPOX	9.000	353.724	81.500	2,572.306
5	TPOX	11.000	361.781	81.500	2,570.156
6	TH01	7.000	191.054	81.500	2,616.112
7	TH01	6.000	186.893	81.500	2,617.242
8	SE33	29.200	408.321	81.500	2,557.769
9	SE33	28.200	404.254	81.500	2,558.849
10	FGA	22.000	260.011	81.500	2,597.452
11	FGA	21.000	255.942	81.500	2,598.549
12	DYS391	9.000	372.942	81.500	2,567.180
13	DYS391	12.000	385.086	81.500	2,563.946

Table 2: Values of peak height after degradation

4. Now need to choose an amplification efficiency for each locus. This can be chosen by choosing a value from the normal distribution N(0, 0.01). Using the below numpy's random normal distribution library the value has generated.

```
np.random.normal(0,0.01,1)
```

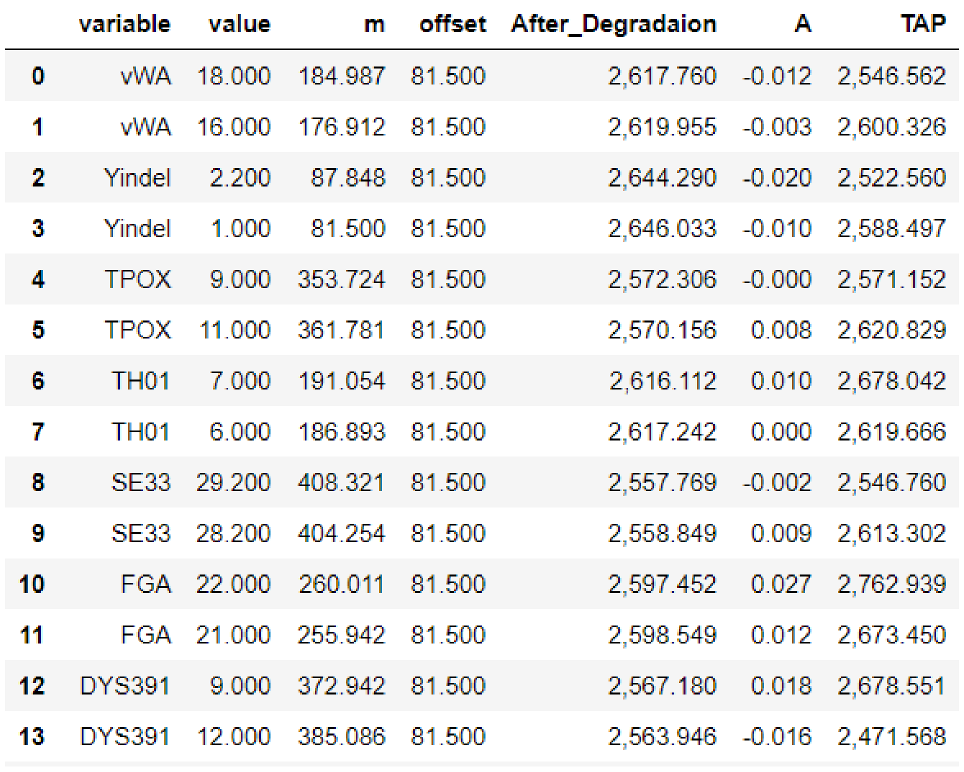
Call amplification efficiency A. The model for amplification efficiency is that each locus is expected to have a log10(A) ~ N(0, 0.01). So to convert the value 'i' from N(0, 0.01) back to an 'A' value that raised 10 to the power of 'i'. Amplification efficiency then affects peak heights by:

```
Peak_height_after_amplification_efficiency =
Peak_height_before_amplification_efficiency * A
```

Now the pre-strutter peak heights (we will call TAP) have calculated by:

TAP = T*A*exp[-D(m-offset)] * X

The values of TAP and A are as follows for just some of the loci shown in table 3:



	variable	value	m	offset	After_Degradaion	A	TAP
0	vWA	18.000	184.987	81.500	2,617.760	-0.012	2,546.562
1	vWA	16.000	176.912	81.500	2,619.955	-0.003	2,600.326
2	Yindel	2.200	87.848	81.500	2,644.290	-0.020	2,522.560
3	Yindel	1.000	81.500	81.500	2,646.033	-0.010	2,588.497
4	TPOX	9.000	353.724	81.500	2,572.306	-0.000	2,571.152
5	TPOX	11.000	361.781	81.500	2,570.156	0.008	2,620.829
6	TH01	7.000	191.054	81.500	2,616.112	0.010	2,678.042
7	TH01	6.000	186.893	81.500	2,617.242	0.000	2,619.666
8	SE33	29.200	408.321	81.500	2,557.769	-0.002	2,546.760
9	SE33	28.200	404.254	81.500	2,558.849	0.009	2,613.302
10	FGA	22.000	260.011	81.500	2,597.452	0.027	2,762.939
11	FGA	21.000	255.942	81.500	2,598.549	0.012	2,673.450
12	DYS391	9.000	372.942	81.500	2,567.180	0.018	2,678.551
13	DYS391	12.000	385.086	81.500	2,563.946	-0.016	2,471.568

Table 3: Values of TAP and A

5. Now need to add stutters to the generated profile. We will need to take into account two types of stutter, one that adds a small peak one allelic position less than the allele (called back stutter) and one that adds a peak one allelic position more than the allele (called forward stutter). Each allele at each locus has an expected stutter ratio (that is the ratio of the stutter peak to its parent peak height). The stutter ratios are given in the databases generated by GlobalFiler.

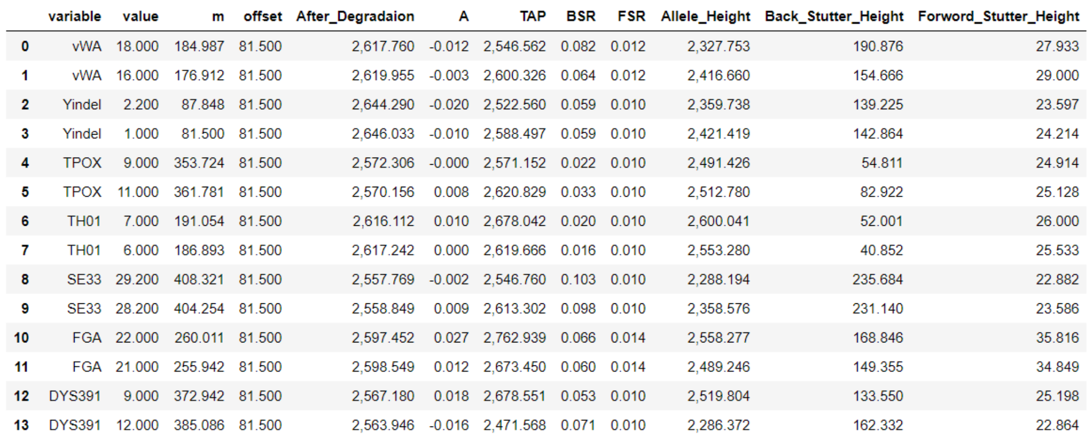
   For back stutter, first check to see whether there is an expected stutter ratio in the GlobalFiler_Stutter_Exceptions_3500 database. This file lists out all observed alleles and locus combinations and their stutter ratio. If this file has a value of 0 for the stutter ratio then move on to the file GlobalFiler_Back_Stutter database, which gives the linear regression parameters that approximately convert allele to stutter ratio. For forward stutter there is only the regression database named GlobalFiler_Forward_Stutter, so just use that straight off. Once the SR for each locus has been obtained (both back stutter ratio BSR and forward stutter ratio FSR) then each TAP height has produced a back-stutter height, a forward stutter height and an allelic height by the below equations:

Allele height = TAP /(1+BSR+FSR)

Back stutter height = BSR* Allele height

Forward stutter height = FSR*Allele height

The values of Allele heights, Back Stutter Heights (BSH) and Forward Stutter Heights (FSH) are as follows for just some of the loci shown in table 4:



	variable	value	m	offset	After_Degradaion	A	TAP	BSR	FSR	Allele_Height	Back_Stutter_Height	Forword_Stutter_Height
0	vWA	18.000	184.987	81.500	2,617.760	-0.012	2,546.562	0.082	0.012	2,327.753	190.876	27.933
1	vWA	16.000	176.912	81.500	2,619.955	-0.003	2,600.326	0.064	0.012	2,416.660	154.666	29.000
2	Yindel	2.200	87.848	81.500	2,644.290	-0.020	2,522.560	0.059	0.010	2,359.738	139.225	23.597
3	Yindel	1.000	81.500	81.500	2,646.033	-0.010	2,588.497	0.059	0.010	2,421.419	142.864	24.214
4	TPOX	9.000	353.724	81.500	2,572.306	-0.000	2,571.152	0.022	0.010	2,491.426	54.811	24.914
5	TPOX	11.000	361.781	81.500	2,570.156	0.008	2,620.829	0.033	0.010	2,512.780	82.922	25.128
6	TH01	7.000	191.054	81.500	2,616.112	0.010	2,678.042	0.020	0.010	2,600.041	52.001	26.000
7	TH01	6.000	186.893	81.500	2,617.242	0.000	2,619.666	0.016	0.010	2,553.280	40.852	25.533
8	SE33	29.200	408.321	81.500	2,557.769	-0.002	2,546.760	0.103	0.010	2,288.194	235.684	22.882
9	SE33	28.200	404.254	81.500	2,558.849	0.009	2,613.302	0.098	0.010	2,358.576	231.140	23.586
10	FGA	22.000	260.011	81.500	2,597.452	0.027	2,762.939	0.066	0.014	2,558.277	168.846	35.816
11	FGA	21.000	255.942	81.500	2,598.549	0.012	2,673.450	0.060	0.014	2,489.246	149.355	34.849
12	DYS391	9.000	372.942	81.500	2,567.180	0.018	2,678.551	0.053	0.010	2,519.804	133.550	25.198
13	DYS391	12.000	385.086	81.500	2,563.946	-0.016	2,471.568	0.071	0.010	2,286.372	162.332	22.864

Table 4: Values of Allele heights, Back Stutter Heights (BSH) and Forward Stutter Heights (FSH)

6. The final height modification has been generated by add some noise (called stochastic effects in forensic science) to the peak heights. The amount that observed peak heights (call these O) vary from their expected height (call these E, and are what have just calculated in step 5) is modelled by:

Log10(O/E) ~ N(0,SQRT(c/[b/E_A + E_A]))

Where c is a 'variability constant' and E_A is the expected parent allele height and b is another constant that limits variability and has a value of 1000 for globalfiler. For allele, back stutter and forward stutter the value of c is 10.34, 9.77 and 85.55 respectively. So, this time added some stochastic effect to a back stutter peak which has an expected height of E_S = 100rfu and its parent peak has an expected height of E_A = 2000rfu. First draw a value 'y' from N(0,sqrt(10.34/[1000/2000 + 2000]) then apply it by:

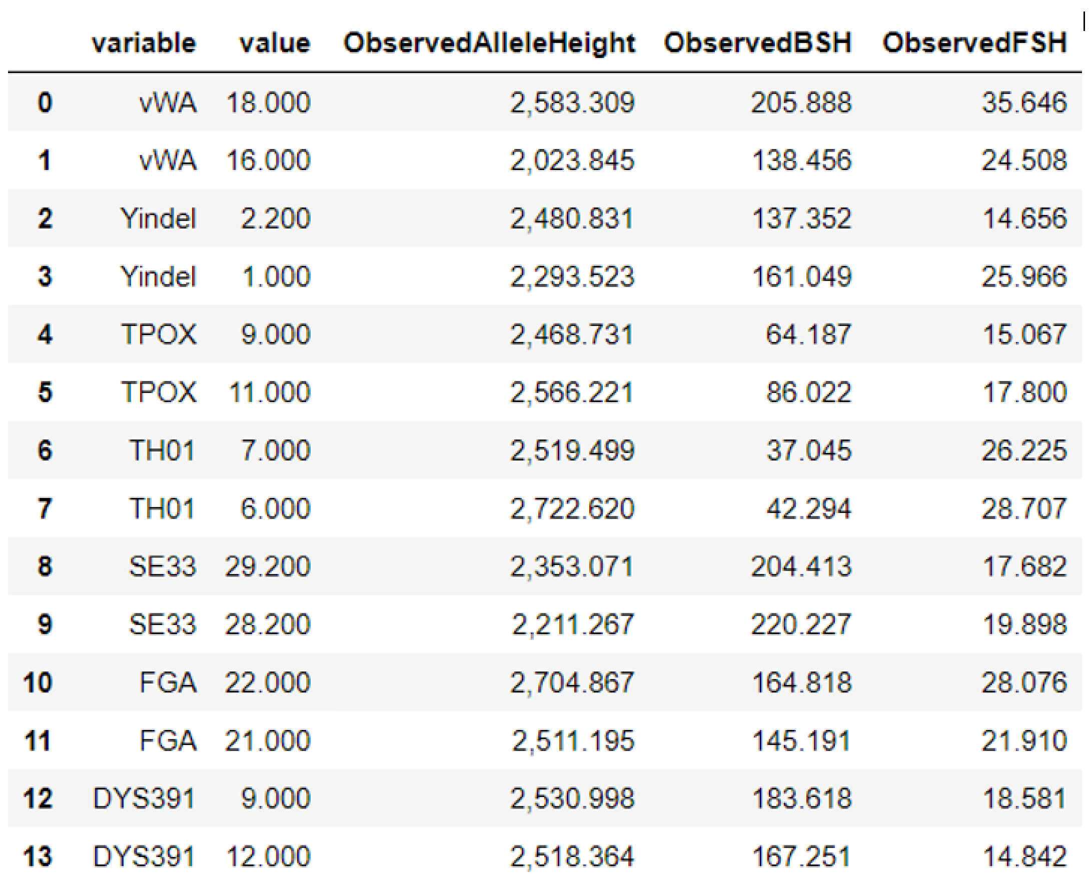
O = E_S *10^y

For each peak, new values for y are drawn. And this procedure has applied on all back stutter, forward stutter and allele peaks obtained in step 5 to get the observed peaks heights.

Below are the scripts that used to generate the values of observed allele heights:

```
tempHeight_y = offsetData.apply(lambda row:
np.random.normal(0,math.sqrt(10.34/((1000/row.Allele_Heig
ht)+row.Allele_Height)),1), axis=1)
ObserverHeightAL = offsetData.apply(lambda row:
row.Allele_Height * 10**row.tempHeight_y, axis=1)
```

The values of observed allele heights, back stutter and forward stutter peak heights are as follows:



	variable	value	ObservedAlleleHeight	ObservedBSH	ObservedFSH
0	vWA	18.000	2,583.309	205.888	35.646
1	vWA	16.000	2,023.845	138.456	24.508
2	Yindel	2.200	2,480.831	137.352	14.656
3	Yindel	1.000	2,293.523	161.049	25.966
4	TPOX	9.000	2,468.731	64.187	15.067
5	TPOX	11.000	2,566.221	86.022	17.800
6	TH01	7.000	2,519.499	37.045	26.225
7	TH01	6.000	2,722.620	42.294	28.707
8	SE33	29.200	2,353.071	204.413	17.682
9	SE33	28.200	2,211.267	220.227	19.898
10	FGA	22.000	2,704.867	164.818	28.076
11	FGA	21.000	2,511.195	145.191	21.910
12	DYS391	9.000	2,530.998	183.618	18.581
13	DYS391	12.000	2,518.364	167.251	14.842

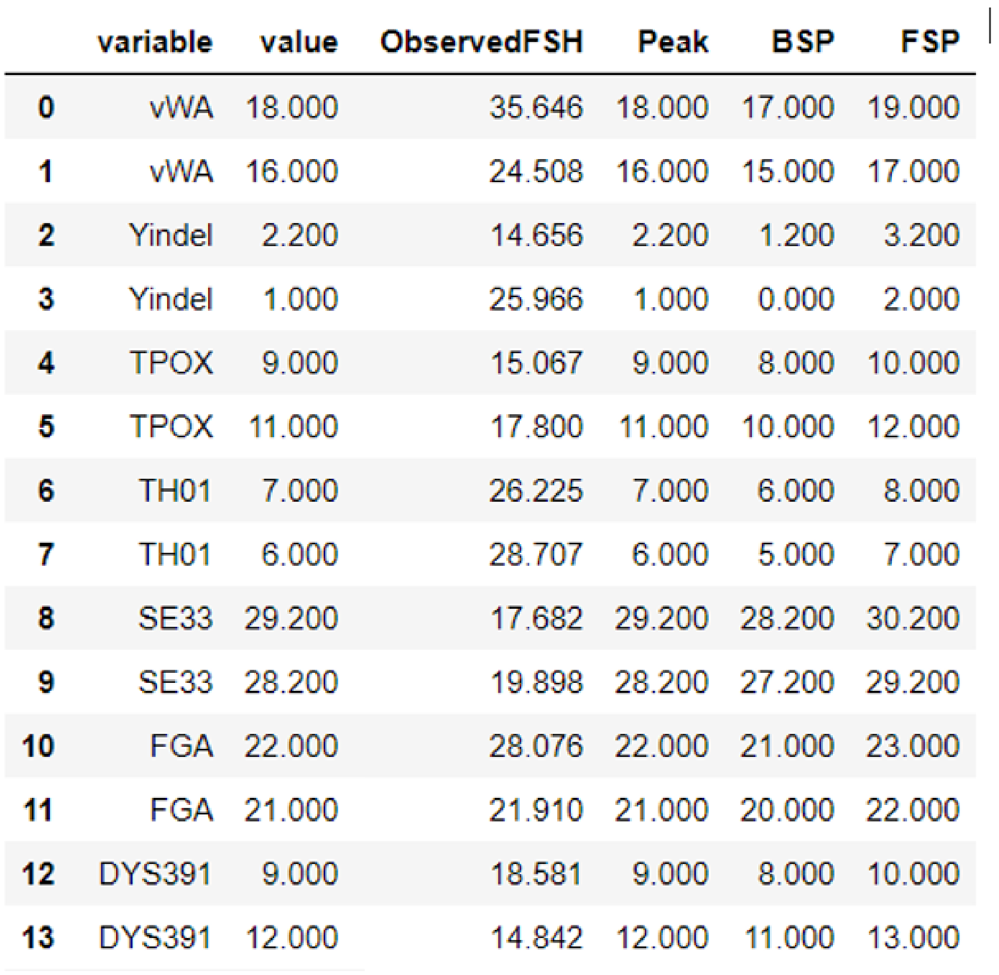
Table 5: Values of Observed allele heights, Back stutter (ObservedBSH) and Forward stutter peak heights (ObservedFSH)

7. The final step is to add any coincident peaks within a locus e.g. if the alleles [10,11] started with [10,11] then it has ended up with:

A 9 back stutter from the 10 allele, a 10 allele and an 11 forward stutter from the 10 allele.

Also, a 10 back stutter from the 11 allele, an 11 allele and a 12 forward stutter from the 11 allele.

In the DNA profile if only observe a 9, 10, 11 and 12 so need to add the two instances of 10 to get the final height of the 10 peak and add the two instances of 11 to get the final 11 peak height. After applied this method in the implementation, I got the values like below:



	variable	value	ObservedFSH	Peak	BSP	FSP
0	vWA	18.000	35.646	18.000	17.000	19.000
1	vWA	16.000	24.508	16.000	15.000	17.000
2	Yindel	2.200	14.656	2.200	1.200	3.200
3	Yindel	1.000	25.966	1.000	0.000	2.000
4	TPOX	9.000	15.067	9.000	8.000	10.000
5	TPOX	11.000	17.800	11.000	10.000	12.000
6	TH01	7.000	26.225	7.000	6.000	8.000
7	TH01	6.000	28.707	6.000	5.000	7.000
8	SE33	29.200	17.682	29.200	28.200	30.200
9	SE33	28.200	19.898	28.200	27.200	29.200
10	FGA	22.000	28.076	22.000	21.000	23.000
11	FGA	21.000	21.910	21.000	20.000	22.000
12	DYS391	9.000	18.581	9.000	8.000	10.000
13	DYS391	12.000	14.842	12.000	11.000	13.000

Table 6: Values of coincident peaks

And the heights after the final calculations be like as follows:

Locus	**Allele**	**Height**
vWA	15	138.456
vWA	16	2,023.845
vWA	17	230.396
vWA	18	2,583.309

vWA	19	35.646

Table 7: Heights of vWA

5.2 Pre-processing DNA Profiles Data to Train Neural Networks:

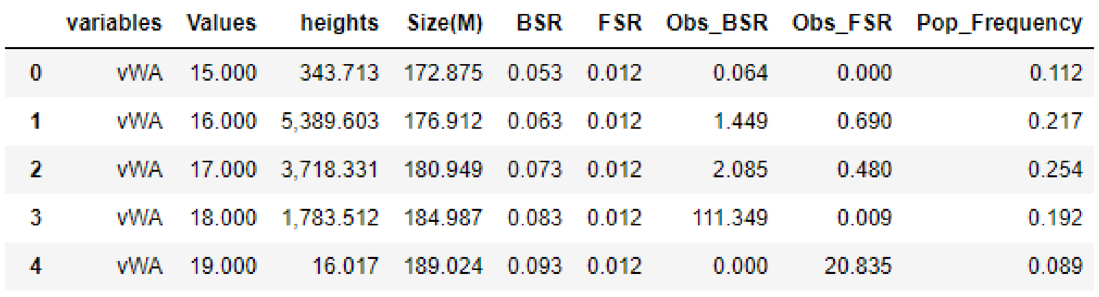
1. The inputs created here for a mixture are on a per locus basis. i.e., for each locus in each mixture the inputs are as follows. In input there are total seven columns for each allele. For example, below are the properties of a generated profile.

Locus	Allele	Height	Size
D3S1358	13	137	113.37
D3S1358	14	1527	117.43
D3S1358	15	3392	121.48
D3S1358	16	2954	125.41
D3S1358	17	87	129.62
D3S1358	18	866	133.61

Table 8: Properties of a generated profile (D3S1358)

- Peak 1 (i.e. allele number: 13).
- Peak height 1 (137).
- Peak size 1 (113.37)
- Expected Back Stutter Ratio (0.054664448 (i.e. from "GlobalFiler_Stutter_Exceptions_3500.csv)
- Observed ratio of peak to peak one allele higher (this is the Observed Back Stutter Ratio (137/1527 =0.089718))
- Expected Forward Stutter Ratio (0.00538 (i.e. from "GlobalFiler_Forward_Stutter_3500.txt"))
- Observed ratio of peak to peak one allele lower (this is the Observed Forward Stutter Ratio (137/0 (This is undefined because there is no observed height for 12, but just put in 0)))
- Peak 1 population allele frequency (From population database)

This mixed DNA profile will be a single input of neural network. These inputs are categorised by the locus. There are total 22 locus exists in a DNA profile (for mixed an individual). After done the above processes on DNA mixture. The inputs are like as follows:



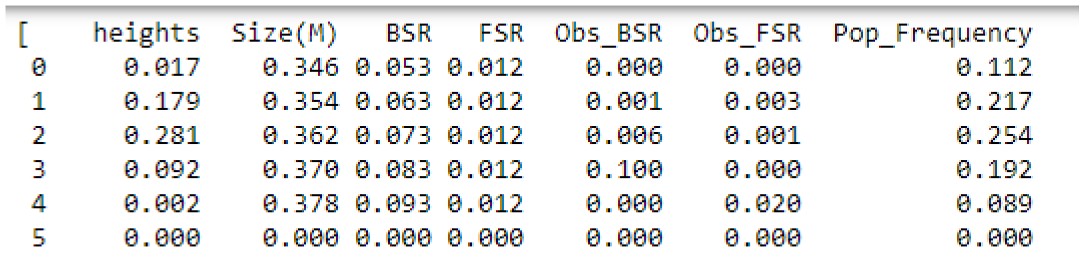
	variables	Values	heights	Size(M)	BSR	FSR	Obs_BSR	Obs_FSR	Pop_Frequency
0	vWA	15.000	343.713	172.875	0.053	0.012	0.064	0.000	0.112
1	vWA	16.000	5,389.603	176.912	0.063	0.012	1.449	0.690	0.217
2	vWA	17.000	3,718.331	180.949	0.073	0.012	2.085	0.480	0.254
3	vWA	18.000	1,783.512	184.987	0.083	0.012	111.349	0.009	0.192
4	vWA	19.000	16.017	189.024	0.093	0.012	0.000	20.835	0.089

Table 9: Input for a single locus vWA (Mixture of 3 individual profile)

In this stage, I have normalised the heights and sizes columns of input data. I did this by considering highest value (X max) 30000 and 500 respectively, being the upper limits for these values and applied on the below equation:

$X' = (X - X \min) / (X \max - X \min)$

By doing this operation on input data, now all values are in the range of [0,1]. And negative values replaced by normalized value.



```
[    heights   Size(M)   BSR   FSR  Obs_BSR  Obs_FSR  Pop_Frequency
 0     0.017     0.346 0.053 0.012    0.000    0.000          0.112
 1     0.179     0.354 0.063 0.012    0.001    0.003          0.217
 2     0.281     0.362 0.073 0.012    0.006    0.001          0.254
 3     0.092     0.370 0.083 0.012    0.100    0.000          0.192
 4     0.002     0.378 0.093 0.012    0.000    0.020          0.089
 5     0.000     0.000 0.000 0.000    0.000    0.000          0.000
```

Table 10: Normalized value of Input for a single locus vWA (Mixture of 3 individual profile)

2. Each [Nx7] input vector contains 3 [1xN] output vectors (These 3 output vectors represented 3 individual DNA profile information). Each individual contributor profile contains two allele number. At vWA there are 9 alleles with a non-zero frequency (this is from your population frequency file, which are using to choose the alleles from). These are 13, 14, 15, 16, 17, 18, 19, 20 and 21. I have assigned assign these alleles positions 1 to 9 in an output array, i.e.

   13 = position 1

   14 = 2

   15 = 3

   ....

   21 = 9

so that to specify a 16, I have put a 1 in the array position 4. For example, a [16,18] will be specified by the output array [0,0,0,1,0,1,0,0,0].

Actual output file contains 3 [1x9] vectors (These 3 output vectors represented 3 individual DNA profile information for locus: vWA).

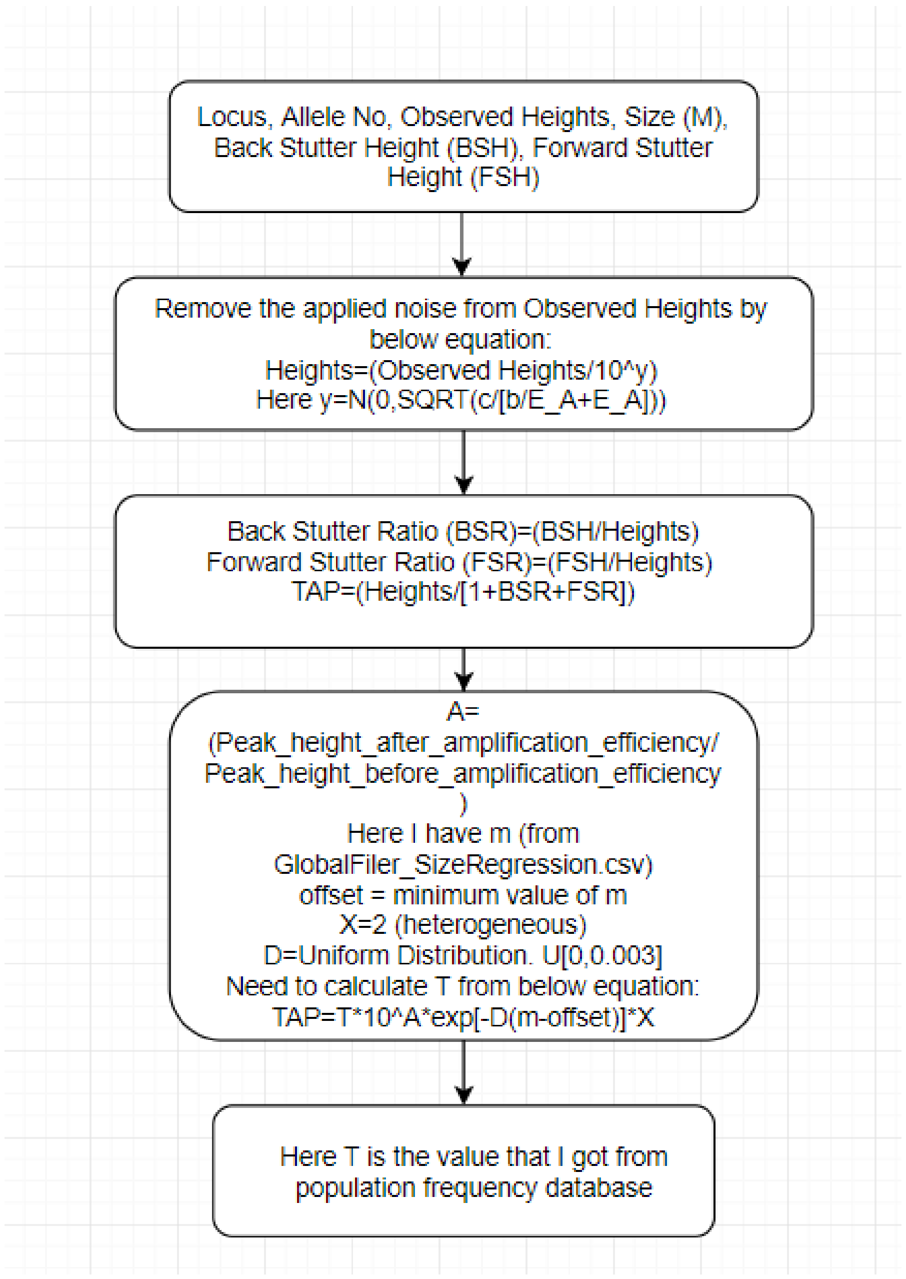
0	0	0	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0
0	0	0	1	1	0	0	0	0

Table 11: Output values for a single locus vWA (3 individual profile)

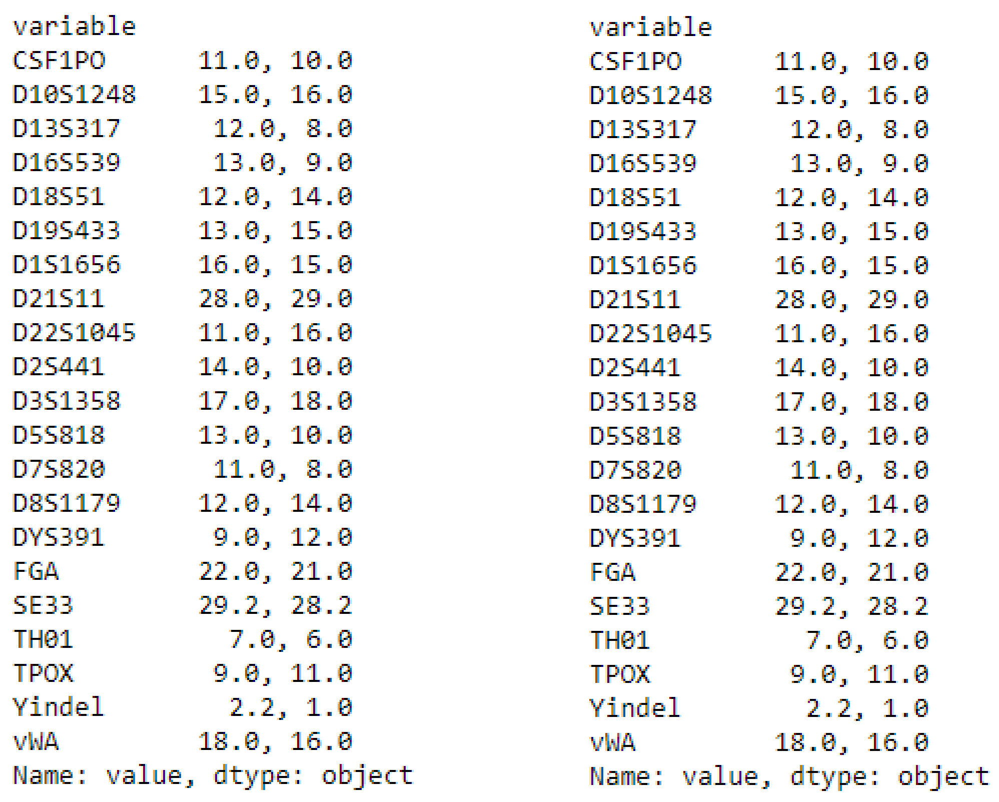
3. Principal Component Analysis (PCA) have applied on dataset to check the performance. PCA is statistical technique that used to reduce the dimension of training data. Though mixed profiles contain the large number of features and the primary issue was high dimensionality. That could cause model overfitting issue. However, after applied the PCA, it was removed some essential parameters. To solve this issue, I have divided the dataset into 21 parts. Each part represents single locus.

5.3 Validation of Simulated DNA Profiles:

In this section, validation processes have been discussed to check produced dataset are valid or not. To do this few dataset samples from generated DNA profiles have taken and applied the below steps on it. The aim of this section is to get the same frequencies back again as in the population frequency database.


Locus, Allele No, Observed Heights, Size (M), Back Stutter Height (BSH), Forward Stutter Height (FSH)
Remove the applied noise from Observed Heights by below equation: Heights=(Observed Heights/10^y) Here y=N(0,SQRT(c/[b/E_A+E_A]))
Back Stutter Ratio (BSR)=(BSH/Heights) Forward Stutter Ratio (FSR)=(FSH/Heights) TAP=(Heights/[1+BSR+FSR])
A= (Peak_height_after_amplification_efficiency/ Peak_height_before_amplification_efficiency) Here I have m (from GlobalFiler_SizeRegression.csv) offset = minimum value of m X=2 (heterogeneous) D=Uniform Distribution. U[0,0.003] Need to calculate T from below equation: TAP=T*10^A*exp[-D(m-offset)]*X
Here T is the value that I got from population frequency database
Figure 5: Steps to validate the simulated DNA profiles

From the above steps, the T values have been determined for the chosen allele numbers.



```
variable
CSF1PO       11.0, 10.0
D10S1248     15.0, 16.0
D13S317       12.0, 8.0
D16S539       13.0, 9.0
D18S51       12.0, 14.0
D19S433      13.0, 15.0
D1S1656      16.0, 15.0
D21S11       28.0, 29.0
D22S1045     11.0, 16.0
D2S441       14.0, 10.0
D3S1358      17.0, 18.0
D5S818       13.0, 10.0
D7S820        11.0, 8.0
D8S1179      12.0, 14.0
DYS391        9.0, 12.0
FGA          22.0, 21.0
SE33         29.2, 28.2
TH01           7.0, 6.0
TPOX          9.0, 11.0
Yindel         2.2, 1.0
vWA          18.0, 16.0
Name: value, dtype: object
```

```
variable
CSF1PO       11.0, 10.0
D10S1248     15.0, 16.0
D13S317       12.0, 8.0
D16S539       13.0, 9.0
D18S51       12.0, 14.0
D19S433      13.0, 15.0
D1S1656      16.0, 15.0
D21S11       28.0, 29.0
D22S1045     11.0, 16.0
D2S441       14.0, 10.0
D3S1358      17.0, 18.0
D5S818       13.0, 10.0
D7S820        11.0, 8.0
D8S1179      12.0, 14.0
DYS391        9.0, 12.0
FGA          22.0, 21.0
SE33         29.2, 28.2
TH01           7.0, 6.0
TPOX          9.0, 11.0
Yindel         2.2, 1.0
vWA          18.0, 16.0
Name: value, dtype: object
```

Figure 6: Comparison of Allele numbers for all 21 loci for a single dataset

5.4 Peak Identification through CNN:

As discussed in previous section, Through Convolution Neural Network (CNN) peak values (allele numbers) have been identified. Allele numbers for individual profiles are declared in output files. These allele numbers for each locus described the number of contributors and their description in DNA mixture profiles.

Actual motive of this section is to identify the peak or allele numbers for all 21 loci of each individual contributors from the mixed profile parameters.

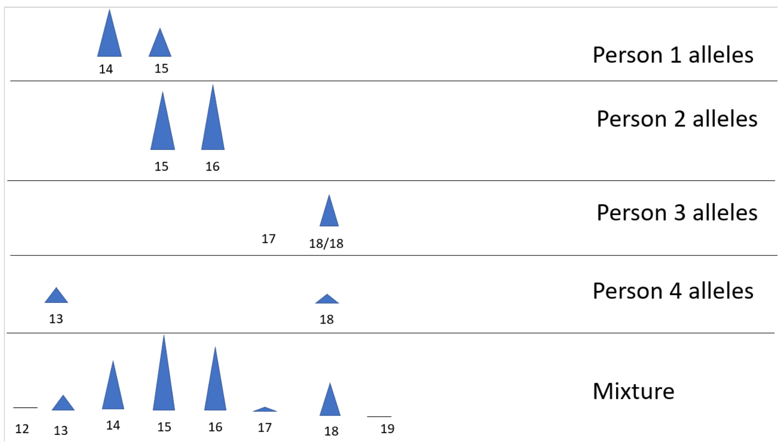


Figure 7: Example of allele numbers of individual profiles and Mixed profile.

In the implementation of Neural Network model, below framework and programming language version have been used.

Language: Python 3.7

Framework: TensorFlow

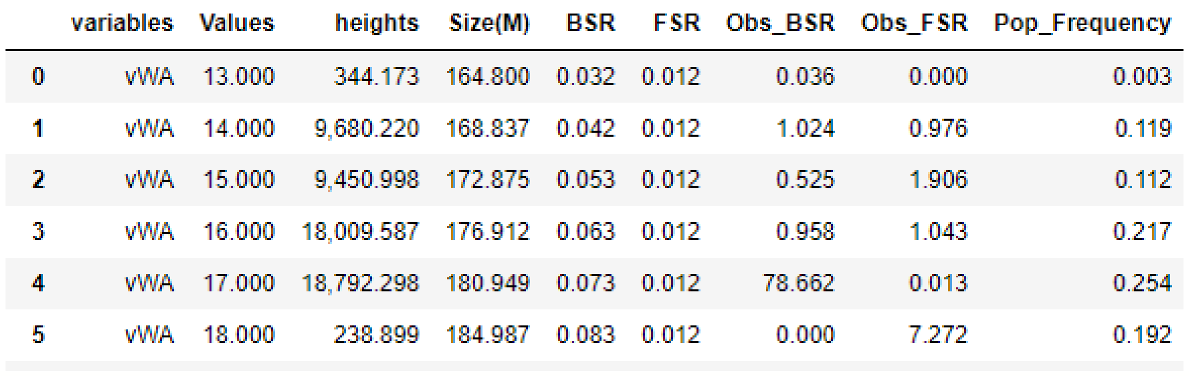
Python Distributor: Anaconda

The following steps have been implemented to get the inputs and outputs, train the neural network models and evaluate the performance of the proposed model. Read data, Normalized the data, Convert to Numpy array, Classification of output data (One hot encoding), Train the data and test, prepare the data for Keras model, prepare sequential data model, flatten all the layers prior train, Choose number of neurons, compare the activation functions (between Sigmoid and Softmax), compile and fitting the model (Optimizer: adam and loss: binary_crossentropy).

```
import os
import tensorflow as tf
import numpy as np

from google.colab import drive
drive.mount('/content/drive')
Dir = '/content/drive/My Drive/22'

def get_file_name(directory, file_name_prefix, count):
  return '%s/%s_%d.csv' % (directory, file_name_prefix, count)

def read_data(path, filename_prefix, num_files):
  X = []
  for i in range(1, num_files + 1):
    file_name = get_file_name(path, filename_prefix, i)
    x = np.genfromtxt(file_name, delimiter=',')
    X.append(x)
  return X
num_files = 2700
X = read_data(Dir + '/input', '0inputvWA', num_files)
Y = read_data(Dir + '/output', '20outputvWA', num_files)
```

Figure 8: Loading Data program in Python

Get_fine_name in using to get input and output files from google colab data directory. And read_data method is using here get the data from all input and output files and append them for further processing. Here for vWA the input file name starts with 0inputvWA_ and output file name starts with 20outputvWA_.



	variables	Values	heights	Size(M)	BSR	FSR	Obs_BSR	Obs_FSR	Pop_Frequency
0	vWA	13.000	344.173	164.800	0.032	0.012	0.036	0.000	0.003
1	vWA	14.000	9,680.220	168.837	0.042	0.012	1.024	0.976	0.119
2	vWA	15.000	9,450.998	172.875	0.053	0.012	0.525	1.906	0.112
3	vWA	16.000	18,009.587	176.912	0.063	0.012	0.958	1.043	0.217
4	vWA	17.000	18,792.298	180.949	0.073	0.012	78.662	0.013	0.254
5	vWA	18.000	238.899	184.987	0.083	0.012	0.000	7.272	0.192

Table 12: Sample of input data before Normalization

In neural network, it is important to normalize the input data. Otherwise, the network prioritizes the large values of input. The actual goal of normalization is to change data to a common scale without changing the value of input data. In the input data file, “heights” column contains the large values (it could be between 100 to 50000), “size” column contains the value between 50 to 500. Here in input files, two columns contain exceptional value. That are “Observed back stutter ratio” and “Observed forward stutter ratio”. In each input file, these two columns contain a single large value compare to other values of these columns. When an outlier detection approach applied on the input data, these large values were deleted as abnormal distance from other values.

```python
def normalize_input(X):
  for k in range(len(X[0][0])):
    mn = 1000000
    mx = -1000000
    for i in range(len(X)):
      for j in range(len(X[0])):
        mn = min(mn, X[i][j][k])
        mx = max(mx, X[i][j][k])
    for i in range(len(X)):
      for j in range(len(X[0])):
        X[i][j][k] = (X[i][j][k] - mn)/(mx - mn)
  return X
X_tmp = normalize_input(X)
# print(X_tmp[:3])
X = X_tmp
# print(X)
X = normalize_input(X)
```

Figure 9: Normalize Input Data program in Python

All input parameters have normalized before train the model. Highest value of a column considered max value to do the normalization. As a part of Neural Network, MaxPooling2D had applied on input data without doing normalization. And in that case only “height” columns were dominating instead of proper modelling. Because “height” columns were containing highest values.

```
X = np.array(X)
Y = np.array(Y)
Y = np.array(hot_vector_2d_to_1d(Y))
```

Figure 10: Data conversion to array program in Python

In this step all input and output file data have converted to NumPy array for further processing. For mixture profiles of three individuals, all output file contains [3x9] vectors for vWA locus, [3x7] vectors for TPOX and so on. And to feed neural network it is needed to convert output file to 1D vectors. Y (or output) cannot be a 2D array and converting it to one hot vectors of size 27 (for locus vWA), size 21 (for locus TPOX) and so on.

```
train_size = 2500
test_size = num_files - train_size
x_train = X[:train_size]
x_test = X[train_size:]
y_train = Y[:train_size]
y_test = Y[train_size:]
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1],
x_train.shape[2], 1)
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1], x_te
st.shape[2], 1
input_shape = (x_train.shape[1], x_train.shape[2], x_train.sha
pe[3])
print(x_train.shape)
print(x_test.shape)

print(y_train.shape)
print(y_test.shape)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

Figure 11: Loading input data to Neural Network Program in Python

Prior build, compile and fitting the model, it is needed to declare the number of training and test size of data. Here in the implementation, 1500 sets of input and output files considered as training size and 100 sets of input and out files have considered as test size. And making sure that the values are float so that we can get decimal points after division. Now reshaping the array to 4-dims so that it can work with the keras API. I would just reshape the input data of vWA locus from (1600, 6, 7) to (1500, 6, 7, 1). And for other locus this shape will be different than others. X_train is a tuple with 3 elements. On the other hand, we know that the input dimensions of vWA are 6x7. So, the shape object's 1st and 2nd element's value is 6, 7. And lastly make sure that the values are float. So that we can get decimal points after division.

```
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Dropout, Fl
atten, MaxPooling2D
model = Sequential()
model.add(Flatten())
model.add(Dense(128, activation = tf.nn.relu))
model.add(Dropout(0.2))
model.add(Dense(y_train.shape[1], activation='sigmoid'))
```

Figure 12: Build the model Program in Python

There are two ways to build a keras model i.e. Sequential model and Functional model. Most of the ConvNets work with sequential model. The sequential model allows to create the models by layer. After that Flatten the 2D arrays for fully connected layers. In Dense layer I have defined 128 neurons for first hidden layer. And assigned the activation function relu and Sigmoid. Relu is sort of transformation that works with non-linearity. Also, it deals with negative values array input array. Relu keeps the non-negative value unchanged and replaces the negative values with 0. On the other hand, Dropout layers fight with the overfitting by disregarding some of the neurons while training. So, here we configure dropout value 0.2 (20%). Dropout can be used during training the models but not during evaluating the performance of model. In this section, Softmax and Sigmoid functions have compared to use in the model. Sigmoid function used as activation function that is used in binary classification. On the other hand, Softmax function also used as activation function that deals with multiple classification logistic regression model. Softmax doesn't work here. Softmax works for increasing score of a label. Sigmoid works on the output between [0,1].

```
Model.compile(optimizer = 'adam',
                    loss = 'binary_crossentropy')
model.fit(x = x_train, y = y_train, epochs = 10)
```

Figure 13: Compiling and fitting the Model Program in Python

‘sparse_categorical_crossentropy' doesn't help in this compile model. Since it is a binary class prediction model. So, ‘adam’ optimizer with ‘binary_crossentropy’ as loss function have been using in the model. In addition, epochs are defined to 10. That means one complete dataset have been trained to neural networks for ten times.

```
Epoch 1/10
1500/1500 [==============================] - 1s 34ms/sample - loss: 0.8191
Epoch 2/10
1500/1500 [==============================] - 0s 266us/sample - loss: 0.8946
Epoch 3/10
1500/1500 [==============================] - 0s 373us/sample - loss: 0.8781
Epoch 4/10
1500/1500 [==============================] - 0s 66us/sample - loss: 0.8519
Epoch 5/10
1500/1500 [==============================] - 0s 182us/sample - loss: 0.8342
Epoch 6/10
1500/1500 [==============================] - 0s 139us/sample - loss: 0.8206
Epoch 7/10
1500/1500 [==============================] - 0s 205us/sample - loss: 0.9031
Epoch 8/10
1500/1500 [==============================] - 0s 221us/sample - loss: 0.8858
Epoch 9/10
1500/1500 [==============================] - 0s 199us/sample - loss: 0.8691
Epoch 10/10
1500/1500 [==============================] - 0s 160us/sample - loss: 0.8589
```

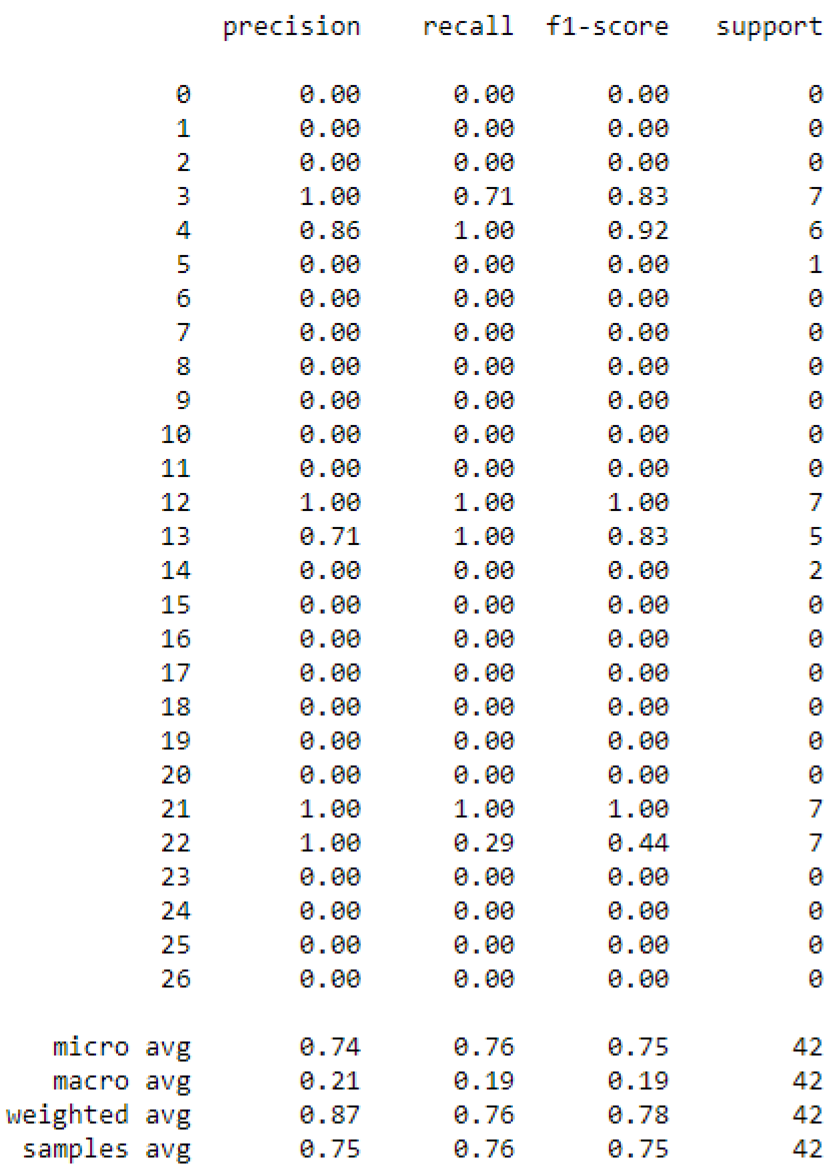
Figure 14: Training performance for epochs 10

5.5 Results and Discussion:

```
preds = model.predict(x_test)
preds[preds>=0.5] = int(1)
preds[preds<0.5] = int(0)

scores = model.evaluate(x_test, y_test, verbose=0)
#print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
#cvscores.append(scores[1] * 100)
#print("%.2f%% (+/- %.2f%%)" % (numpy.mean(cvscores),
numpy.std(cvscores)))
scores

from sklearn.metrics import classification_report
print(classification_report(y_test,preds))

from sklearn.metrics import classification_report
print(classification_report(y_test,preds))

preds = convert_outputs(preds)
print(preds)
y_test = convert_outputs(y_test)
print(y_test)
```

Figure 15: Predict and Evaluating the Model Program in Python

In the first part of the program, rounding up the floating-point values to binary classes based on thresholds. After that converting back the one hot vectors to original 3x9 matrix for vWA locus, 3x7 matrix for TPOX locus etc. It is important to converting back to original output format for manual evaluation of performance. Here the classification report (for precision, recall, f1-score and support) has been generated to discuss about the performance of neural model.



```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.00      0.00      0.00         0
           2       0.00      0.00      0.00         0
           3       1.00      0.71      0.83         7
           4       0.86      1.00      0.92         6
           5       0.00      0.00      0.00         1
           6       0.00      0.00      0.00         0
           7       0.00      0.00      0.00         0
           8       0.00      0.00      0.00         0
           9       0.00      0.00      0.00         0
          10       0.00      0.00      0.00         0
          11       0.00      0.00      0.00         0
          12       1.00      1.00      1.00         7
          13       0.71      1.00      0.83         5
          14       0.00      0.00      0.00         2
          15       0.00      0.00      0.00         0
          16       0.00      0.00      0.00         0
          17       0.00      0.00      0.00         0
          18       0.00      0.00      0.00         0
          19       0.00      0.00      0.00         0
          20       0.00      0.00      0.00         0
          21       1.00      1.00      1.00         7
          22       1.00      0.29      0.44         7
          23       0.00      0.00      0.00         0
          24       0.00      0.00      0.00         0
          25       0.00      0.00      0.00         0
          26       0.00      0.00      0.00         0

   micro avg       0.74      0.76      0.75        42
   macro avg       0.21      0.19      0.19        42
weighted avg       0.87      0.76      0.78        42
 samples avg       0.75      0.76      0.75        42
```

Figure 16: Classification Report of model for vWA

From the classification report, it can be said that higher true positive rate or precision represents the total predictive power of a model. 100% precision is found in class variable 3, 12, 21 and 22 where 3 belongs to the first individual profile, 12 from second individual profile and 21 & 22 from third individual profile. As this model can produce most of the class variables with around 100% accuracy rate for three individual profiles, so it can be said that this model has comparatively good predictive power for DNA profiling. In contrast, total accuracy of the model is around 75% where error rate depicts 25% that means this model can successfully predict 75 times out of 100 occurrences. And overall accuracy of the model has been evaluating by model.evaluate function. Based on the current training the performance is 85.1% (Based on all 21 Loci). On the other hand, based on the classification report, Precision, Recall and F1 Score have been calculated for all 21 loci. Precision has been calculated based on true positive score and false positive score. Recall has been calculated based on true positive and false negative. And F1 score calculated based on Precision and Recall. After that average them to get final evaluation. In addition, micro average, macro average, weighted average and samples average have been calculated as well for comparison. The values of Precision, Recall and F1 Score are listed below for all 21 loci.

Loci	Precision	Recall	F1 Score
CSF1PO	0.82	0.87	0.8442604
D10S1248	0.84	0.89	0.8642775
D13S317	0.89	0.92	0.9047514
D16S539	0.92	0.91	0.9149727
D18S51	0.82	0.83	0.8249697
D19S433	0.86	0.81	0.8342515
D1S1656	0.91	0.92	0.9149727
D21S11	0.88	0.87	0.8749714
D22S1045	0.82	0.81	0.8149693
D2S441	0.82	0.81	0.8149693
D5S818	0.83	0.82	0.8249697
D7S820	0.82	0.84	0.8298795
D8S1179	0.87	0.86	0.8649711
DYS391	0.85	0.84	0.8449704
FGA	0.87	0.86	0.8649711
SE33	0.89	0.88	0.8849718
TH01	0.87	0.88	0.8749714
TPOX	0.81	0.82	0.8149693
Yindel	0.83	0.82	0.8249697
vWA	0.74	0.76	0.7498667
Average	0.848	0.851	0.8493438

Table 13: Neural Network Performance Statistics for all Loci

Above performance measured on total 56700 sets of data with 21 Neural Networks (each NN represents a single locus). Each neural network was trained with 2500 dataset and tested with 200 datasets.

6. Further Development:

Further development of this neural network can be implemented for the mixture of N number of contributors. In this current work, model has been designed for mixture of 3 and 4 individual contributors. However, in further development, number of contributors in a mix profile can be determined. Also, current model is unable to determine the individual profile that contain same number allele. Like: if any locus contains the same number of allele (i.e. 19,19). Current model can be compared with other optimised version of neural networks like: Recurrent Neural Network (RNN). In addition, a desktop software application can be developed that can be able to profile the DNA and able to setup and evaluate the different type of neural networks and other classifiers (like: Rapidminer).

There would be many swabs from around the crime scene reflecting different proportions of contributors' DNA. If there are all up K contributors to a crime scene and we take ≥K swabs with different (possibly 0) weightings of (subsets of) the contributors, it is possible to mathematically (perfectly) solve that for the contributors, their DNA and their weightings. An ELM model would suffice to do this in ANN terms if we knew the original DNA of some or all the contributors. The weighted ranked contributor representation CNN could be a pre-processor for this.

7. Project Deliverables:

The research and development of this Neural Network system will be focused on following deliverables:

1. Develop python code that simulates realistic DNA profiles.
2. Trial a system of CNNs (and potentially other machine learning algorithms) to identify the number of contributors to the simulated DNA profiles.
3. Identify the limitations of performance for determining the number of contributors to any DNA profile.
4. Analysing the machine learning and data classification algorithms for the related classification problems to find out best solution to train the neural networks.
5. Trial a system that takes into account the raw data from the laboratory instrument and process it using existing ANNs as inputs into the ANNs for determining the number of contributors.

The outcomes of the research and development are as follows:

1. Reviewed existing Neural network systems and literature.
2. Finalised the method and algorithms and implementation of the DNA profiling part.
3. Collect and generated dataset and DNA samples.
4. Validate the generated dataset.
5. Complete the prototype of the Neural Network system.
6. Running simulation to test the system are working properly.
7. Finalising documentation based on all final finding.

8. Conclusion:

There are lots of manual tasks involved in the analysis of DNA in the field of forensic science. Lot of challenges are involved here including quality of sample (low profile DNA), quality of samples can be issue, also the manual calculation that involved in the examination. In addition, number of contributors from a single mix swabs identification is a challenging task. There are lot of potential scopes in this field of research. The project has developed to find a solution using neural network that can interpret the individual contributor from DNA mixture. In the implementation, DNA profiling has done for single contributor and developed the EPG analysis graph for it. After that convolutional neural network (CNN) has been imposed on the DNA mixtures for different number of contributors to explore the possibility of identifying the allelic peaks. From the classification and performance evaluation report, it can be said that higher true positive rate or precision represents the total predictive power of a model. 100% precision is found in maximum class variable. As this model can produce most of the class variables with around 100% accuracy rate for three individual profiles, so it can be said that this model has comparatively good predictive power for DNA profiling. In contrast, total accuracy of the model is around 75% where error rate depicts 25% that means this model can successfully predict 75 times out of 100 occurrences. So far with this model, Accuracy is over 85% for all 21 loci and it can be increased by increasing the number of test profiles. On the other hand, average precision for all 21 loci is 84.8% and average recall for all 21 is 85.1%.

Further development of this neural network can be implemented for the mixture of N number of contributors. In this current work, model has been designed for mixture of 3 and 4 individual contributors. However, in further development, number of contributors in a mix profile can be determined. Also, current model can be compared with other optimised version of neural networks like: Recurrent Neural Network (RNN).

9. References:

[1] D. Taylor, J. Bright and J. Buckleton, "The interpretation of single source and mixed DNA profiles", Forensic Science International: Genetics, vol. 7, no. 5, pp. 516-528, 2013. Available: 10.1016/j.fsigen.2013.05.011.

[2] M. Marciano and J. Adelman, "PACE: Probabilistic Assessment for Contributor Estimation— A machine learning-based assessment of the number of contributors in DNA mixtures", Forensic Science International: Genetics, vol. 27, pp. 82-91, 2017. Available: 10.1016/j.fsigen.2016.11.006.

[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, 2019. Available: 10.1016/j.fsigen.2018.10.019.

[4] M. Woldegebriel, A. van Asten, A. Kloosterman and G. Vivó-Truyols, "Probabilistic peak detection in CE-LIF for STR DNA typing", ELECTROPHORESIS, vol. 38, no. 13-14, pp. 1713-1723, 2017. Available: 10.1002/elps.201600550.

[5] D. Taylor and D. Powers, "Teaching artificial intelligence to read electropherograms", Forensic Science International: Genetics, vol. 25, pp. 10-18, 2016. Available: 10.1016/j.fsigen.2016.07.013.

[6] C. Lawless, "The low template DNA profiling controversy: Biolegality and boundary work among forensic scientists", Social Studies of Science, vol. 43, no. 2, pp. 191-214, 2012. Available: 10.1177/0306312712465665.

[7] R. Cowell, T. Graversen, S. Lauritzen and J. Mortera, "Analysis of forensic DNA mixtures with artefacts", Journal of the Royal Statistical Society: Series C (Applied Statistics), vol. 64, no. 1, pp. 1-48, 2014. Available: 10.1111/rssc.12071.

[8] A. Dakhli, W. Bellil and C. Amar, "Wavelet Neural Network Initialization Using LTS for DNA Sequence Classification", Advanced Concepts for Intelligent Vision Systems, pp. 661-673, 2016. Available: 10.1007/978-3-319-48680-2_58.

[9] W. Cheng, J. Huang and C. Liou, "Segmentation of DNA using simple recurrent neural network", Knowledge-Based Systems, vol. 26, pp. 271-280, 2012. Available: 10.1016/j.knosys.2011.09.001.

[10] W. Zang, X. Liu and W. Bi, "An Artificial Neural Network Classification Model Based on DNA Computing", Human Centered Computing, pp. 880-889, 2015. Available: 10.1007/978-3-319-15554-8_82.

[11] "Recurrent neural networks 101: Understanding the basics of RNNs and LSTM", *Built In*, 2020. [Online]. Available: https://builtin.com/data-science/recurrent-neural-networks-and-lstm.

[12] W. Cheng, J. Huang and C. Liou, "Segmentation of DNA using simple recurrent neural network", Knowledge-Based Systems, vol. 26, pp. 271-280, 2012. Available: 10.1016/j.knosys.2011.09.001.

[13] G. Lo Bosco and M. Di Gangi, "Deep Learning Architectures for DNA Sequence Classification", Fuzzy Logic and Soft Computing Applications, pp. 162-171, 2017. Available: 10.1007/978-3-319-52962-2_14.

[14] H. Ling, S. Samarasinghe and D. Kulasiri, "Novel recurrent neural network for modelling biological networks: Oscillatory p53 interaction dynamics", Biosystems, vol. 114, no. 3, pp. 191-205, 2013. Available: 10.1016/j.biosystems.2013.08.004.

[15] P. Chen, L. Chang and F. Chang, "Reinforced recurrent neural networks for multi-step-ahead flood forecasts", Journal of Hydrology, vol. 497, pp. 71-79, 2013. Available: 10.1016/j.jhydrol.2013.05.038.

[16] K. Raza and M. Alam, "Recurrent neural network based hybrid model for reconstructing gene regulatory network", Computational Biology and Chemistry, vol. 64, pp. 322-334, 2016. Available: 10.1016/j.compbiolchem.2016.08.002.

[17] K. Liu, P. Zhong, Y. Zheng, K. Yang and M. Liu, "P_VggNet: A convolutional neural network (CNN) with pixel-based attention map", PLOS ONE, vol. 13, no. 12, p. e0208497, 2018. Available: 10.1371/journal.pone.0208497.

[18] D. Quang and X. Xie, "DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences", Nucleic Acids Research, vol. 44, no. 11, pp. e107-e107, 2016. Available: 10.1093/nar/gkw226.

[19] X. Du, Y. Yao, Y. Diao, H. Zhu, Y. Zhang and S. Li, "DeepSS: Exploring Splice Site Motif Through Convolutional Neural Network Directly From DNA Sequence", IEEE Access, vol. 6, pp. 32958-32978, 2018. Available: 10.1109/access.2018.2848847.

[20] A. Deshpande, "A Beginner's Guide To Understanding Convolutional Neural Networks", Adeshpande3.github.io, 2020. [Online]. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/.

[21] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition", Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, 1998. Available: 10.1109/5.726791.

[22] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997. Available: 10.1162/neco.1997.9.8.1735.

[23] D. Rumelhart, G. Hinton and R. Williams, "Learning representations by back-propagating errors", Nature, vol. 323, no. 6088, pp. 533-536, 1986. Available: 10.1038/323533a0.

[24] Phung and Rhee, "A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets",

Applied Sciences, vol. 9, no. 21, p. 4500, 2019. Available: 10.3390/app9214500 [Accessed 23 June 2020].