Developing an Android Application for Outfit Recommendations by Implementing Machine Learning Techniques

By Joseph Mckeown

University of Reading

Department of Computer Science

Developing an Android Application for Outfit Recommendations by Implementing Machine Learning Techniques

By Joseph Mckeown

Supervisor: Xia Hong

*Module:*CS3IP16-20 -Individual Project

Date of Submission: 28/04/2021

A report submitted in partial fulfilment of the requirements of the
University of Reading for the degree of
Bachelor of Science in *Computer Science*

Declaration

Abstract

Neural networks are used to solve complicated problem with large amount of data by being able find the trends in the data provides. They aim to solve the problem by learning the relationships within the data which can be used for multiple functions such as classification but also prediction. It has historically been computationally taxing to run but due to the modern improvement in processing speed it has become much easier to implement a neural network into other applications. The project creates an android application which is provides the service to which recommends outfit to the user. It has combined this core function with an integrated neural network to have the recommendation be based of the taste of the user that the network has learned. The application has 3 main screens each with a core element of functionality with item recommendation, item editing and item adding. This project proves that this application can learn the taste of a user and can provide better suggestions over time. The result of the project is an application which can be downloaded to android device where the user can add representation of their clothes and have the app give them recommendation based on their clothes.

Table of Contents

Chapter 1 - Introduction

One of the most common tasks which every person faces each day is to choose what to wear. This is an important decision as what you choose to wear has an influence on both your mood and how other see you. [1] Because of this a lot of time being spent choosing what kind of outfits to wear. This is time consuming process as there are many factors to consider. This can be the synergy one item of clothing has another, how the colours work with each other or taking into consideration the practical issues such as the weather too got. Some applications have tried to solve this problem by creating wardrobe planners. These are apps where the user can plan an outfits day by day. With the recent developments in the neural network and recommender systems, a lot of these decision-based problems like this being able to be solved by these systems. This presents a unique opportunity to implement this technology to help solve the problem of what to wear. This has been showed to be success as more and more companies use this technology themselves. By using it they have seen higher success rates than before. This is shown in how shopping sites such as amazon, who used these systems, have led to a 29% annual sales increase which these systems greatly contributed to [2].

This project aims fill the gap in market for a smart outfit application by develop its own. The goal is to create an Android application which can accurately recommend outfits based on a user's tastes. It aims to understand how modern neural networks are made and to adapt them for the purpose of learning tastes of clothes. The reason for this is so it can make better predictions for outfits. The project wants to use neural network techniques to discover the trends form a user's decision to wear certain clothes. It hopes and uncover the trends about their outfits to be able to give better recommendations.

The motivation was to develop a smart solution to the problem of what to wear. The potential of neural networks was able fuel the idea of creating an innovative solution. This is because of a neural networks capability to be able to distinguish relationships in data made it an ideal feature to implement. This was a great way to innovate the decision-making process. The goal of this is to be able to create an increased level of user satisfaction from using the application from this feature. Along with this, it would be a distinct selling point to the application compared to the apps competing in the same market space.

This report will discuss the key steps which were taken to develop this application. It will firstly discuss the results of the research into the technologies and systems which are implemented in the app. It will also review how these technologies can be to develop the app. Then the methodology of how the application would be created would be outlined. It shows how the researched neural networks would be implemented and translated to the app. As well the designs on how the app will look and function are defined here. From this the report will show how the designs were successfully implemented into an application. The results of testing the application are displayed proving that it can fulfil the goal of learning the tastes. Finally, a summary of the output of the development process will be discussed and how this may be expanded upon in future works.

Chapter 2 - Literature Review

2.1 - History of Neural Networks

The development of neural network has had theories written about if for many decades form as early as 1900's. This stemmed from the concept of trying to create an artificial representation of the human brain, replicating how human process thoughts and their ability to learn. One of the earliest discoveries was in the 1943 with the development of the McCulloch-Pits (MCP) Neuron model.[3] This model was the basic design of an artificial neuron based of a biological model. In the biological model of the brain, the dendrite receives signals from other neurons. These signals are then processed by the soma. If the signal received is strong enough from the dendrite, exceeding its threshold, it will send a signal through the axon which either is excitatory or inhibitory. Excitatory signals encourage an action being made while inhibitory signals deters from the action [4] This model was translated to create the MCP neuron. This is where numeric inputs are received by the computer as with a dendrite. If the sum of the inputs exceeds the threshold for the neuron, it would send a representation of an excretory or inhibitory signal being a 1 or 0. This development of a basic model of neuron was to be able to learn certain Boolean logic problems such as And, Or and Not.

This model expanded upon further with developments such as the perceptron in 1958 from Frank Rosenblatt [5]. This helped expand the fundamental ideas about neural networks by introducing weighting into the model. This is where each input into a neuron has a weight associated with it. This weight affects how strong the input is. Once run, the weights of the neuron based on the error, being the difference between the true answer and the predicted answer. This facilitates a way for the neuron to be able to learn the problems. Because of this, the model is able to work on linearly solvable classification problems.

However, issues arose, as even though a perceptron could learn certain problems it would struggle with others which were not linearly separable. This could be problems such as the XOR problem. This would then be solved by the later development of the Multi-Layer Perceptron. A Multi-Layer-Perceptron is a model where is another layer of perceptrons between the input and output nodes. This was allowed for non-linearly separable problems.

To help with training these networks, multiple method were invented. One of the most popular and prevalent ones is back propagation which was first proposed by Paul Werbos. This is the process of changing the weights in a Multi-Layer-Perceptron based of the error of the next layer and the connection strength.

2.2 - Modern Deep Learning Systems

In recent years, as the processing power of computers has increased, artificial neural networks have become far more prevalent topic of research along with their greater implementation in applications. This is brought by the deep learning. This is where an unsupervised multi-layered neural network model is fed huge amounts of data. From the data, it can distinguish trends and relationship within the data to learn the problem. Unsupervised networks are given information to learn without any labels to train against. This means there is no human intervention in learning the data. The deep neural network discovers the trends in the data and finds which features are important for each classification.[6] The structure of the network contains multiple layers of neurons which are called the hidden layers. This is where the majority of the learning occurs. Often with more layers the better the performance will be.

Due the success of deep neural networks of being able to learn vast amounts of data with no human interaction, it has been implemented far more commercially. This has seen implementation in such fields such as voice recognition systems being used in application such as Siri and Amazon Alexa[7]. It is also become able to learn more complicated problems such as image recognition and manipulation where the implementations are constantly improving. This is seen with applications such as Facesawp being one of the leading open-source version of deepfake software [8].

2.3 - Exploration of Recommender systems

Recommender systems which is a subclass of information filtering which predicts the preference of the user. These are usually things the user wants, and the goal is to be able to predict these from the users' habits. "RS are used primarily for individuals who lack sufficient personal experience or competence to evaluate all the choices." [9]. Recommender system often basses its prediction based on the actions the users takes or the known history.

A recommender system is defined by the users' preferences and constraints. By knowing this information, recommendations can be made which conform to each user's preference. There are multiple way data is collected to fuel recommender systems. This can be explicit like their reviews or they can be interpreted from a user's action or though associated likes. Some of these techniques are used in modern example such as Facebook with their collaborative filtering system which creates ratings for recommendations based of people with similar interests [10]. With using metrics based on a user's action it could be the about of retention a user has on a certain site, video or article. The goal for recommender systems is to filter all the options provided and find the best options to display based on each user. This is done as it predicts the rank a predicted item and comparing it to other possible options.

An effective method for recommender systems is called collaborative-filtering. This is uses comparisons of other users who have similar tastes. The idea is that if one user likes the same thing as other users, there is a higher chance that the one user may like some of the things the other users like.[11] The similarity in tastes with other users would increase the rank of an item making it more likely to be shown. This has been shown to be an effective method as many companies like Amazon use method like this in how they recommend products to its users. This type of filtering has become much easier to make more accurate recommendations. This is due to the amount of data which is harvested from users. This means there is even more data than ever before for the recommenders.

There many benefits by implementing a recommendation system for both the user but also the owner. For the user, it provides the ability to have a better selection of choices while the owner benefits as it often mean greater profit. This can be due to the user finding something they want to buy, or the customer experience being improved leading to longer use of the application/site and establishing consumer loyalty.

Recommendations have become more and more important in modern life as due to increased ease of access to information though the internet. With recommender systems, it streamlines the decision process and reduces the amount of choice having to be made. This has the effect of being able to help reduce decision fatigue. This being caused by the overwhelming about of choice which is facilitated. Decision fatigue is the phenomenon where the quality of decisions become worse the more decision which must be made throughout the day [12].

2.4 - Modern Implications of Artificial Neural Networks in Recommender systems

Recently these two techniques have been intertwined more often. This is due how deep learning can process large amounts of data and be able to recognise trends in the data while unsupervised. This benefits recommender systems as it provides a way to produce far more accurate predictions for recommendation. The style of network used for this is called Neural Collaborative Filtering. This uses multi-layer perceptron networks to learn the interactions with the users. Though using networks like this, the performance of a recommender systems is improved significantly. This is because it can alleviate shortcomings of traditional recommender system techniques such as matrix factorisation. Neural Collaborative Filtering has benefits over traditional methods as due to its nature of a deep neural network based model, it is able to learn problems which contain more non linearly solvable relationships which other method would struggle or miss the connection.[13]

This system has been adopted into a lot of modern company's recommender systems. One such example is YouTube which uses data obtained from their Tensor flow brain to train their deep neural network. [14] As google have a wealth of data from it large userbase and Tensor flow, this means that their recommender models trained through Neural Collaborative Filtering will have process more accurate results. This would also produce results of a higher accuracy what collaborative filtering would have.

2.5 -Competing fashion applications

When researching for applications which are in the same field as the project, the majority of fashion application focused on outfit were outfit planners. These are applications which allow the user to create a virtual wardrobe filled with their clothes. They then use this wardrobe to create outfit and save them either as ideas or to be used for certain days. This is a very common approach to solving the problem of what to wear. They have more of a focus organisational aspects of the concept as well as the social media aspects.

One of the biggest examples of this type of app is Combyne.[15] With 149,848 user reviews and an average review score of 4.5, it is one of the most popular fashion and outfit related apps on the google play store. This app creates an environment to add clothes in the form of picture. This picture are cropped so that they can be demoed on a virtual mannequins. This gives the users a visually appealing way to display their outfits. The focus of the app is the ability to test outfit as well as being able to share the outfit ideas with other users. As well as this, it boasts integration with online outlets by linking users to the retailer's site when discovering a item of clothing form another outfit. This application is strong example of the general outfit application as it's extra feature such as sharing of outfit and retail links give it a unique edge. An inspiration this project took away form this is the visualisation of the outfit themselves being a appealing way to display them.

Another out app is Shuffle Outfits.[16] This is an app which has more in common with the idea of this project. This is because this app's core functionality is to give the users to ability to a virtual wardrobe to randomise random outfits. This is a way for any combination of outfits be shown potentially keeping outfits fresh for the user. However it is a fairly basic function as all it does is choose an outfit randomly. The way it created an outfit is by defining an outfit as comprising of shoes, something for the lower half of the body such as trousers and something for the top half such as a hoodie.

2.6 - Application to the project

The project's goal is to be able to create an outfit recommender application with the unique feature of being able to learns the user's tastes. To achieve this goal, the application will need to take advantage the methods mentioned previously to learn this problem. Implementing a Neural Network will provide a way to learn the tastes. To do this, a multi-Layer Network would be ideal as this kind of problem would as it would allow for deeper learning of the users tastes. Along with this implementing some aspect of deep learning would help boost the performance of the application as well as uncover trends about the user's clothes.

Another way is to implement recommender system techniques in deciding on the prediction. This would be intertwined with the neural network as the recommender would base it's decision based off the results of the neural network as seen with google. Implementing these systems should help to fulfil the goal of satisfying the user as these systems will improve the recommendation given.

Form exploring other application in the same field it has shown that a smart system for recommending outfits will give the project an innovative edge. In the design of the application the project has taken inspiration form some of it’s more relevant aspects. Combyne’s method of visualizing the outfit has been adapted as it provides a good visual reference for the outfits. From Shuffle Outfits, the way it simplifies and clearly defines an outfit will be a useful tool to implement as it will make the selection process much easier.

In summary, the future of neural networks is going toward deep learning-based implementation. An implementation of this is used for the solution to learn the taste in clothes. This will be implemented to inform the decision made by the recommender system to boost it’s accuracy. In the design of the application, it will take inspiration some of the other application in the same field while still implementing a unique spin.

Chapter 3 – Methodology

When designing the application, 3 major parts have to be designed and considered being the neural network to be used, the method for virtualisation of the wardrobe and the design and structure of the application itself. Each of the major aspects of the application will have their ideas, design, and structure to be used discussed and analysed.

3.1 - Design of the Neural Network

The neural network model is the backbone of the application. It is what allows for the core concept of recommendations to function in the application. The aim for the neural network in the app is to be able to learn the user's tastes of clothes. This means considerations must be made to what the structure of model will be. It needs to be able to that network use the clothes as an input. This will be done by having each individual item have its own input along with each outfit having an input into the network as well. This will facilitate the need to create a distinction for the performance of outfits as a whole and items individually. The idea of which is because an individual item of clothing may not be popular with the user on its own but when paired with other specific items the outfit it more favour. Another factor when designing the neural network, the output of the network will need to be processed by a recommendation system to decided which outfit is shown.

The first decision made about the network was what kind of network would be used for the project. Due to the way the network wants to use it's inputs it means that the problem will not be linearly separable. This is because an item of clothing may rate bad individually but may do well in an outfit. This would mean that there would not be able to create a linear relationship between good and outfits. This means that this problem cannot be solved by a single layer perceptron network [17]. As with what was researched earlier a multiple layer network can be used to solve this kind of problem.

For the find the inputs for the network, it finds them from the current item of clothing/outfit being rated. This works by having the current item/s associated input value being set to 1 while all the other items have a value of 0. This method ensures that only the values of the relevant items are used running the network. This is also important in changing the weights, as the network only wants to change the weight of the relevant input’s weights. As well, each input has its own weight attached to it. This affect how strong the signal from the input will be within the network. These weights affect each input individually and are changed throughout usage of the network. The way each weight is affect is either positively or negatively depending on the user decisions. This allows for more favoured outfits to be discovered as the item weights. With a greater input into the network, it will be producing a higher output overall. The design of how the inputs are processed into the network is modelled below:

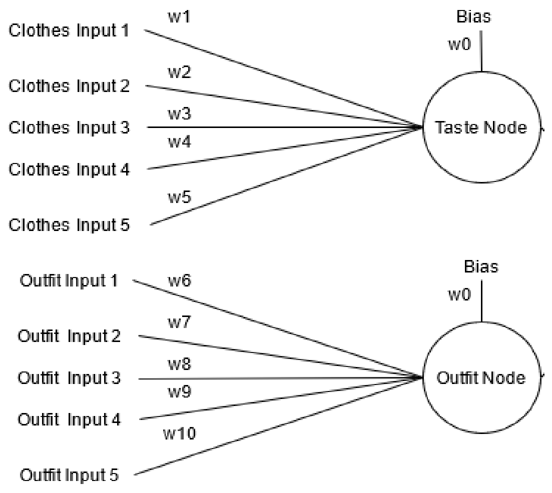


Figure 1 Inputs from the wardrobe into the neural network

As the problem cannot be linearly separated, a multiple layer perceptron need to build from the input nodes. To create this network, there would need be a layer/s of perceptrons between the input layer and the output layer. These layers of perceptron are called the hidden layer. The amount of hidden layer which are include will affect the performance of the network.

For a neural network to learn a problem it needs a goal. This goal is used to decided how the network will learn by basing it off the error from what was desired and what was produced. To facilitate the learning in the network, it would receive the goal based on the user’s decision on an outfit. If they like an outfit and choose yes, the goal will be 1 while if they don’t and choose no the goal will be set to 0. This will be used to change the weight within the network accordingly. Due to the network being a MLP network, the weights are changed through back propagation. Back propagation “calculates the gradient of the error function with respect to the neural network's weights” [18]. This calculates the error of each layer based of the layer before it is starting at the final output layer and working it's way back. The error of the a layer in backpropagation is calculated by multiplying the previous layers error by the connected weight between the two layers This would be described as the chain rule[19]. The goal of this is employ gradient decent to each layer within the network. Gradient decent is when we want the weight in the network to produce the smallest amount of error, as if the overall error is decreasing, we want to increase the weight.

A model displaying the structure of the applications neural network can be made. This help to visualise how the network will function and also how to implement it.

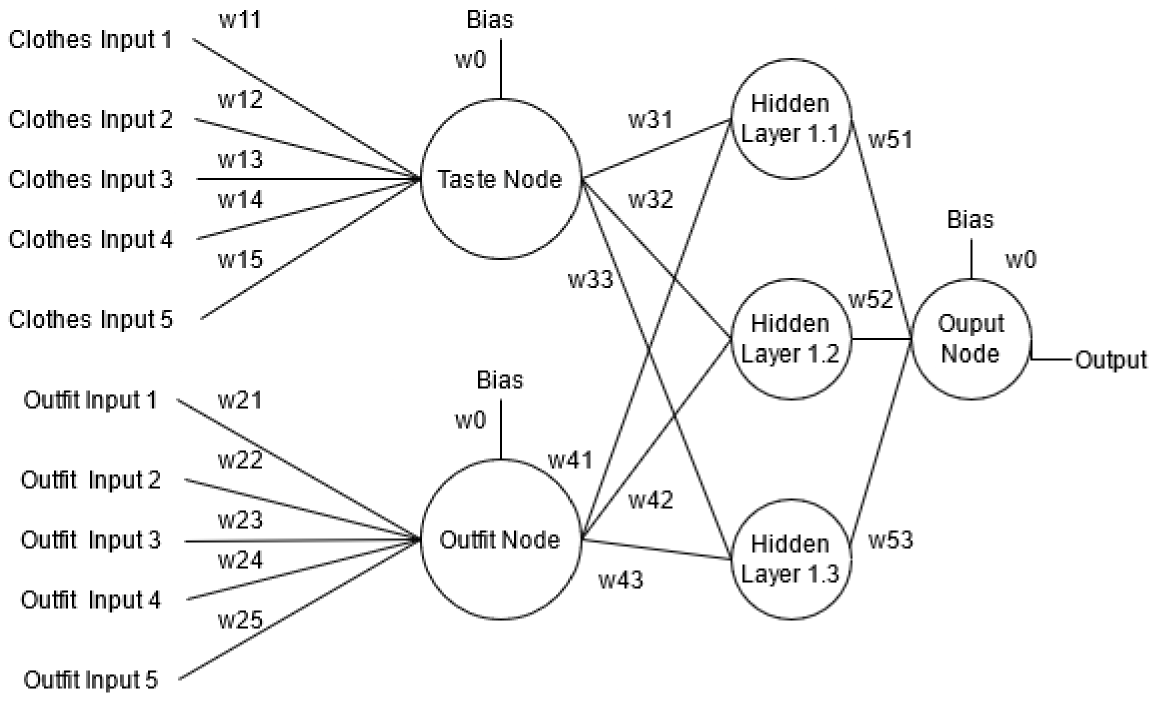


Figure 2 MLP design of the Neural Network with a single hidden layer

As this projects neural network is not using a data sets for training the application it will have to learn the taste solely though the users use of the application. This is different to how other neural networks based in classification, as they would be trained on sample data to learn train the weight of the network. This means that the way the network is set up will have to learn the pattern within a reasonable pace. This means the weight must be fairly sensitive to the results from the user. The rate/speed of learning can be changed by altering the learning rates and the momentum within the network itself. A way in which this can be achieved is by having the network itself set to a higher learning rate. This is because the learning rate of a network dictates how large a change is made to the weights in a network. Generally smaller change is better as it allows for more nuances to be learned. It also helped prevent outliers in data drastically changing the weight and output. This is important as if the network can changes too much due to outlier it could lead to it learning the wrong information about the data or unlearn data. This would affect the performance of the network as it would produce more inaccurate results.

As this type of problem is a prediction, the output of the network would be used to predict what outfit is most likely to be preferred by the user. This means that there need to be selection occurring in the network. To do this it needs all the possible results of the outfits in the network need to be found. From this the output would need to be compared to each other to decided which outfit shall be the predicted outfit. This is where the recommendation system can be built. With its purposes to choose the best outfit for the situation.

The recommender system will take inspiration from the techniques from used in Evolutionary Algorithms. Evolutionary Algorithms are inspired by biological evolution and are often use a similar problem to find the best answer to a problem. This is done through creating a population with potential answers and has the strongest survive each generation. The parts of evolutionary algorithms which will help with making the recommender are Evaluation and Selection. In evaluation, the population is given a score, called fitness, based on how close a chromosome in the population was to solving a problem. This fitness score is then used within selection, where the parents of the next generation are selected based of their fitness and the algorithm's selection technique. Some selection techniques include rank based, where the chromosomes with the best fitness are chosen to reproduce with other high fitness parent, and roulette, where each chromosome is given a proportional percentage chance to be chosen based on the chromosome's fitness. These techniques can be implemented into the recommender as it can use the result the produced in the neural network as the values to evaluate fitness. From this an appropriate selection algorithm can be used to decided which outfit should be chosen as a prediction.

A rank-based selection method could work with recommending as it would always provide the best possible out to the user. However, an issue arises as this method would keep presenting the same outfit if the user keeps liking the outfit. This would mean that there would be low to no variety of outfits being shown. Even though this would always produce the best result, it is also important that the users are presented with variety of outfits.

A method which was experimented with to try alleviating this issue, was to implementing a blacklist. This blacklist would exclude a chosen outfit from participating in the rank selection whilst on the blacklist. This would have worked in a similar manner as dating sites such as Tinder where once a decision has been made on a person they are taken out of the queue. [20]This idea would be adapted so that an outfit would be eligible to re-join the potential outfit after spending enough time there or when the amount of clothes blacklisted exceed a threshold of the overall wardrobe size. Any outfits which would eligible would have a chance to be added back each recommendation. This however would not be a good enough solution as it this could lead to outfit which are well liked being removed from the potential wardrobe for a long time. This could result in lower user satisfaction, as they may not understand why an outfit is not appearing anymore.

In the end, the recommender would use a roulette-based selection method. This is because, it gives outfits a greater preference to be chosen if they have performed well in neural network and vice versa if seen negatively. This would be achieved while still being able to have another outfit be chosen. This makes it a preferable option than a rank-based selection as it will still make preferred outfit have the better chance of being chosen while also allowing for less liked outfit, but still equally viable outfits to be chosen.

Complexity of the wardrobe

Each run of the recommender, it will loop for o times, with o being the amount of outfit. O = i*j*k; i=number of tops, j=number of underTops, k = number of bottoms. This means when ever a new item is added to the wardrobe the complexity is increased by i*j for a k, j*k for a I and i*k for a j. This means the number of outfits increases at a bigger rate the more item of clothing are already in the wardrobe.

3.2 - Design of the Application

The design of the app would need to be able to facilitate the main function of the app. These functions being:

- Neural network recommendation
- Wardrobe management
- Clothes additions / deletions
- Outfit storage

To accommodate this, the app will have separate screens to focus on each task. With this idea, 3 screens would need to be created. It would also need a central screen which would contain the main function of recommendation. As the other elements of functionality compliment the recommender, they would work options to access from the main screen.

<u>Screen1 - Recommender</u>

For this main screen, the user would need to be able to request a recommendation from the current items of clothing available and to present it on screen. It also needs the interactivity for the user to decide if they like or dislike the recommended outfit. This can be done through a having button either side of the recommend button. These would provide each option, respectively. As well, due to it being the main screen of the application it would need to have suitable menu functionality. This can be done by having buttons at each corner at the top of the screen. These can lead to both the clothing item addition screen and the wardrobe management screen. This would follow standard designs where the options and extra feature are accessible in other screens.

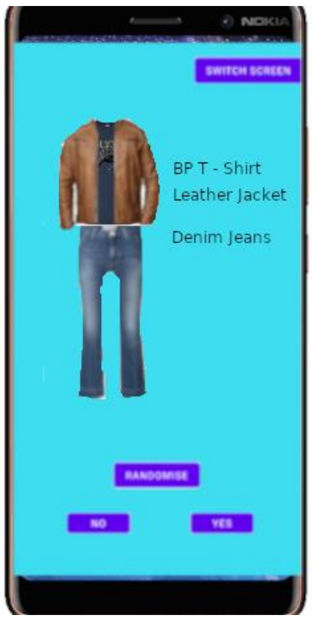


Figure 3 Early Demo of main Screen

In the main screen, the main processes are the recommendations. When the user is using this screen, will follow a general cycle of interactions. This would be them obtaining recommended outfits and deciding whether they like the outfits or not. This would occur with them occasionally adding more cloths to the application. With this in mind a flow chart can be create showing the general usage of the screen.

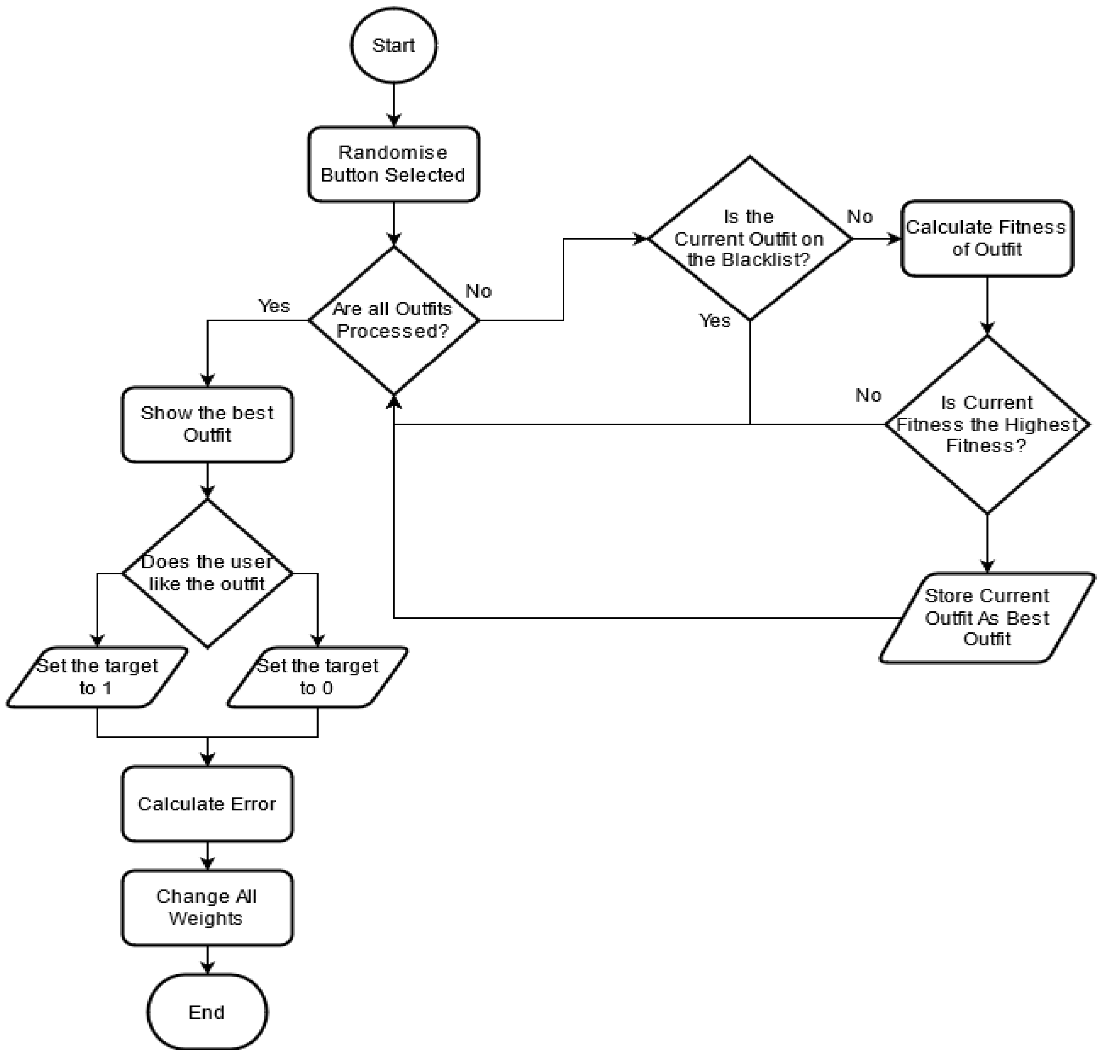

Start
Randomise Button Selected
Are all Outfits Processed?
Yes
No
Is the Current Outfit on the Blacklist?
No
Yes
Calculate Fitness of Outfit
Is Current Fitness the Highest Fitness?
No
Store Current Outfit As Best Outfit
Show the best Outfit
Does the user like the outfit
Set the target to 1
Set the target to 0
Calculate Error
Change All Weights
End

Figure 4 Flowchart of the main Screens main process

Screen2 – Wardrobe Management

In this screen, the user will be able to see the entirety of the clothes which they have entered the application. This screen will allow the user to browse through each item and highlight them to gain more information about the item of clothing. This can be information like an item's current fitness. These kinds of feature can help the users better understand how the application is functioning. This can inform them of which items of clothing which they prefer. This will help towards the goal of a better user experience as it can help them better understand their tastes through this app.

Another feature in this screen is the ability to edit the properties of the items which are in the wardrobe. The user should be able to change the majority of the key information about an object like the name, image representing it, delating the item and even what category it belongs in. As this is changing key information about the item, consideration have to be made for adjusting these values without affecting the neural network. For both the Name and the Images they should be able to be delt with within a Clothes class. This is because they are the properties to define the clothes which are not intertwined with the NN. However, if the user would want to delete or to change which sub category the item belonged in accommodation would need to be made to ensure the NN says accurate. This would be ensuring the weights are removed accordingly with the deletion of an item. As well it would need to remove any outfit which may have contained the item. For the changing which type it belongs to, it would need to have the have all the outfit weights containing that item removed as those outfits would no longer be possible. Along with that it would need to create whole new outfit weights for all the new outfits that this type change would have created.

Screen3 – Adding Clothes

This screen will act as the main hub for the users to be able to add the items from their wardrobe into the application's virtual wardrobe. This screen would need to have the option for the user to input all the relevant information about each item of clothing. This will contain the name of the item, a picture of the item and the sub-type which the item fits under. With all this information it should be ready to add a new item to the virtual wardrobe and the neural network.

To add new item to the neural network, consideration must be made on how it would create the new weights for it. individually, it would need to set up the creation of a new weight. With outfits, all the new outfit created due to the inclusion of this new item would need to be indexed. Along with this the weights would need to be created for the respective outfits which are created. This may take a long time as the with lots new outfits being added it means that there's a larger wardrobe so may take some time to calculate all the outfits as the number of outfits added per item is relevant to the amount of other item in the other categories. This could be resolved by either implementing multi-threading when it is possible on a device or to have an item limit where only a certain about of items can be added where the outfit limit would not be exceeded.

The design of the screen would need to have the needed functionality to facilitate add item. The way it will be done is but providing the user multiple field to interact with on the screen. The image below displays an early design for this screen.

On the screen, the user is provided multiple options:

- 3 buttons which align with the 3 main categories for the clothes in the clothing classifier.
- Textbox for the user to input the name of the item of clothing
- A button which allows the user to take a picture of an item.
- Confirmation button which creates a new item of clothing in the wardrobe

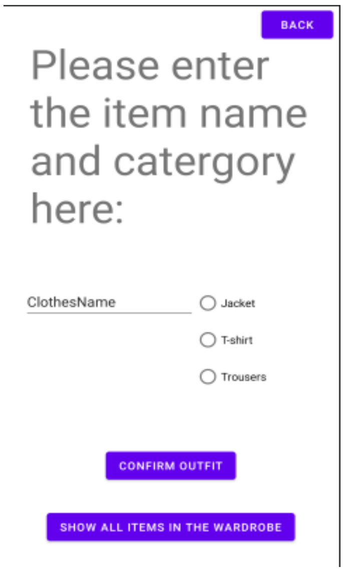

BACK
Please enter the item name and catergory here:
ClothesName
Jacket
T-shirt
Trousers
CONFIRM OUTFIT
SHOW ALL ITEMS IN THE WARDROBE

Figure 5 Mock upl of the item adding activity

With these options, the user is able to engage with adding item to the app. Some considerations need to be made when implementing as it needs to ensure that an invalid item is not entered into the wardrobe. This will be done by ensuring the options have all been filled on confirmation of the Item. This requires that the clothing item to be entered is a valid object. It is checked to see if it has the right attributes being a name, type and a picture. Once validated it will be added to wardrobe. If the requirements for an item clothing in the app was expanded the validation would equally need to be extended.

3.3 - Design of Virtualised Wardrobe

The main objects within the application are the clothes. To represent the clothes within the application a class would be used. This class will contain the multiple parameters needed to define what the item of clothing was. One of the most important parameters is the type of clothing it is. As there are many types of clothes, it needs to simplified to make the description of an outfit easier. It would also help with classifying the item. To achieve this each item of clothing in the application would need to be classified as a type. The type would represent a distinct category of an outfit. An outfit is be represented a one item from each type This is done to prevent items which would be incompatible in real life becoming outfits E.G. not wear 3 pairs of jeans. A way in which this can be executed is by having each of the clothes to be sub-dived into 3 main categories being a Top, Under Top, Bottom. These would represent the basic components of what would make a usual outfit.

By using this clothing classification, it makes it simple and effective to classify clothes and prevent conflicts when creating outfits. It can also allow for further customisation with the user as you could change the minimum and maximum requirements needed for an outfit to be considered. Another way in which it can be expand in is to include more sub-categories. This could be very useful as it gives more specific option for the users to classify their item which will help track their preference in certain subtype of clothes. This can help provide more accurate results in the neural network. This is because it could be influenced by the preferences of more subtypes. This is because the more specific the categories the easier it become to implement category-based filtering/learning. This because the more specific the categories are, the less generalisation must be made if implementing it into the learning process. For example If the user likes jeans but they are classified under long bottoms(E.g. jeans, trousers, leggings) it will recommend others within that sub category even if the user has no affinity for clothe of that type. However for simplicity and ease of implementation the project will use the 3 main categories described before as it's definition of an outfit.

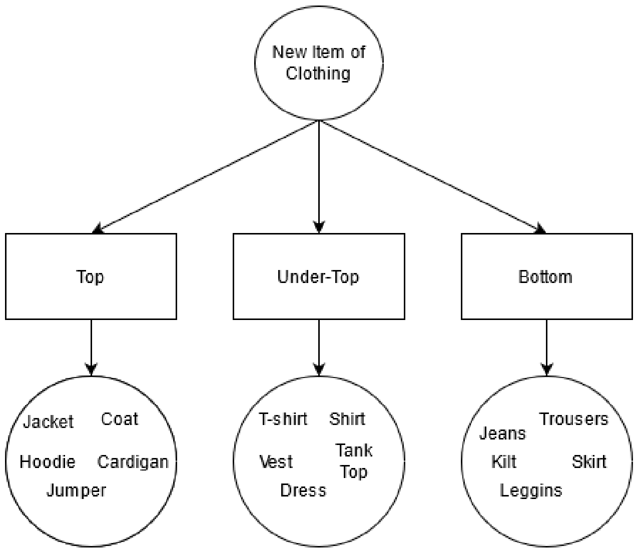


Figure 6 The Design of the Clothes Classification System

Chapter 4 - Implementation

The platform which was decided to create the application was android. This is because the android OS is having the majority market share in smartphone demographic. Because of this, it would allow for the application to be run on more devices, making it easier to distribute.[21] To develop the app itself, android studio, the official IDE for android, was used. This is because the IDE provides a lot of feature such as comprehensive layout editors, virtual device emulators and useful debugging tool, all of which made it appealing to use. Android studio uses both Java and Kotlin as the primary languages.[22]For the application, Java was used. This is because, the developer has more experience using Java, along with it being an object-oriented language which is a necessity for the creation of the application.

Android application are comprised of multiple activities. In each of these the UI can be created from the layout editor which generates an XML file with the UI design. Along with this each activity has an associated java file. This contains any of the functions and process needed to run the activity.

In the application, it is broken down into 3 activities:

- MainActivity
- itemAdding
- WardrobeActivity

These activities each contain one of the main processes of the screens discussed earlier.

Along with the activities, the application also contains multiple separate java files. These classes help to manage all the process for its respective task. These classes are used within the activity classes. The java classes which are in the application are:

- NeuralNet
- Network
- Node
- Clothes

- Outfit

For each of the 3 activities there some universal functions which they all share. These are mostly relevant for the transversal of the application. In each screen there are 1 or 2 traversal buttons in the corners of the screen. As the main Activity is the core of the application the screen goes between the main activity and the other activities. Below shows how each screen is connected to each other.

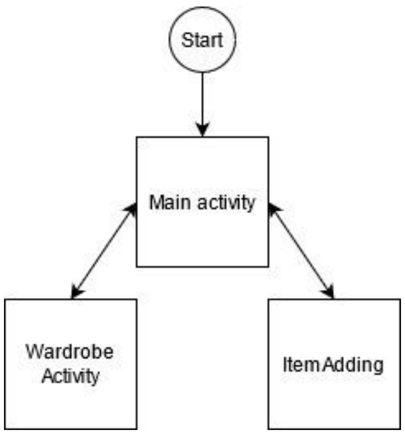
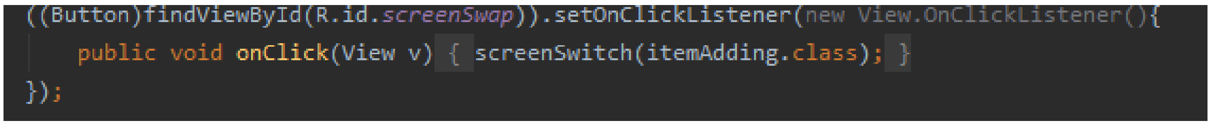


Figure 7 application screen flow

To leave an activity for another, certain information needs to be passed through. This is information which is necessary for the functioning of the app. This means that the Neural Network object needs to be passed from one activity to another. This can be done by adding extras to an intent object. Intent objects are what allow other activities to be run. Along with this each intent can store information in the form of extras. To pass the wardrobe object information form one screen to another, its important information is stored as extras. Once the new activity has started it will extract the intent information sent over and use another constructor to rebuild the NeuralNet object in the new activity. This constructor takes the passed extras and inputs them back into a NeuralNet object. For each of these traversal functions, it implements a try can catch when adding the extra information. This is so the user can traverse though the screens even if there is no information to pass from the neuralNet. Along with that it prevents the application from crashing if there are any issues with the data.

All of these functions are implemented in buttons in each respective screen. Each screen sets the button's onClickListener on its initialisation. It is slightly different for the mainActivity as there are 2 traversal buttons. These button onClickListeners are set so the call the same function but pass a different class object. This allows for the same function to be used for traversing to both screens.



```
((Button)findViewById(R.id.screenSwap)).setOnClickListener(new View.OnClickListener(){
    public void onClick(View v) { screenSwitch(itemAdding.class); }
});
```

Figure 8 Main Activity Screen Swap button onClickListener

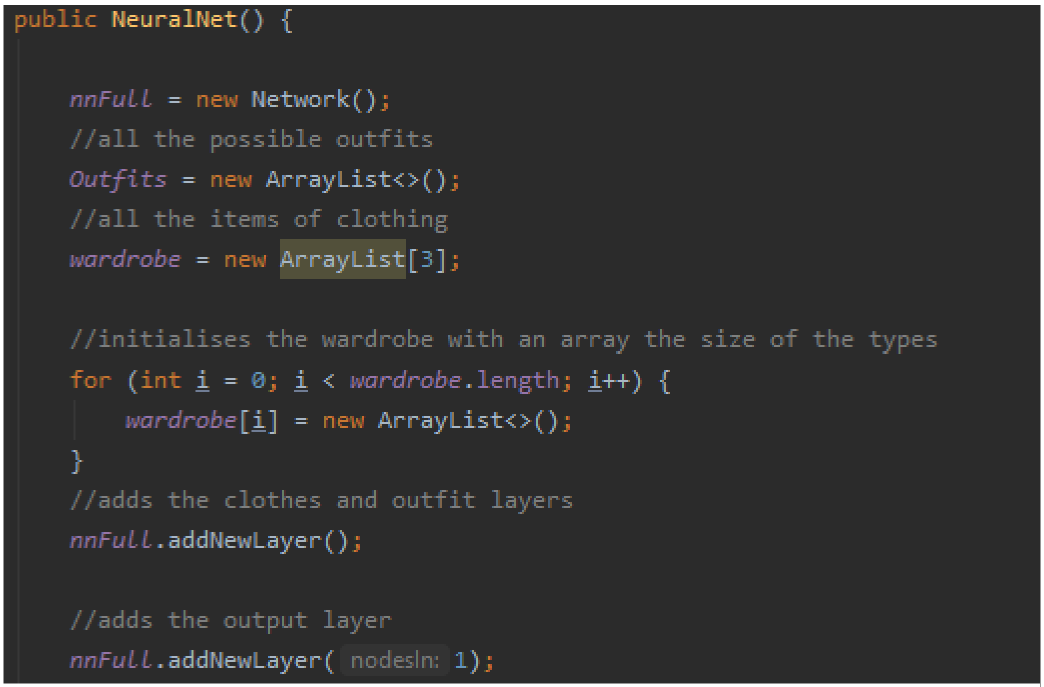
4.1 Class descriptions

<u>NeuralNet class</u>

The NeuralNet class is used to contain and process all the information for a neural net to run in the application. This class links with multiple other classes as it contains multiple object from other classes.

The wardrobe of clothes is stored in this class. The wardrobe is implemented as an ArrayList of Clothes objects. Along with this all the possible outfits are stored in another ArrayList only made of Outfit objects. Both of these array lists dynamically change in its size and contents throughout using the application. This is from new Clothes objects being added to the application and old objects being removed.

The network itself is built from creating a Network object.



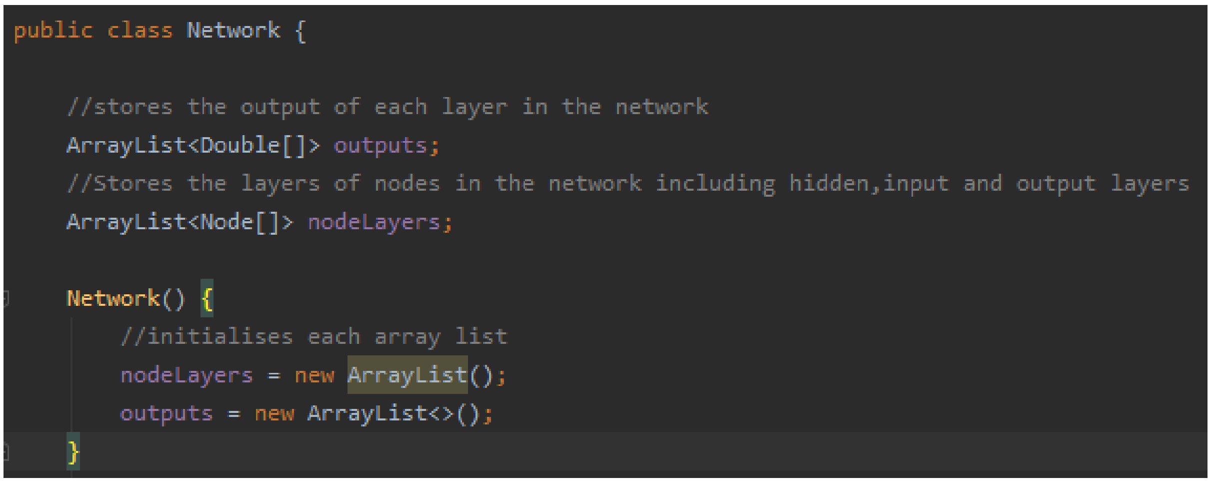
```
public NeuralNet() {

    nnFull = new Network();
    //all the possible outfits
    Outfits = new ArrayList<>();
    //all the items of clothing
    wardrobe = new ArrayList[3];

    //initialises the wardrobe with an array the size of the types
    for (int i = 0; i < wardrobe.length; i++) {
        wardrobe[i] = new ArrayList<>();
    }
    //adds the clothes and outfit layers
    nnFull.addNewLayer();

    //adds the output layer
    nnFull.addNewLayer( nodesIn: 1);
```

Figure 9Neural Network constructor

Network Class

This class contains all the information to be able to create a neural network. It does this by storing an ArrayList of an array of node objects called nodeLayers. By having the node objects being stored in an array format, it can represent multiple layers of the network. For example, to represent an output layer of 2 output nodes, an array of 2 nodes would be added to the list. This makes it easier to apply changes for an entire layer. As well it helps with connecting nodes from one layer to another. Along with that this makes it very easy to implement hidden layers as node layers can be created of a set amount of nodes to a set amount of layers. Also having the layers in this format help greatly with implementing back propagation. This is because it can be implemented into a for loop for the number of layers, working until it reaches the input layer.

The main function of this class is to calculate most of the network related information like setting up the network layers, applying the changes in the network, calculating the output of the network and the relevant getters and setters for the network.



```
public class Network {

    //stores the output of each layer in the network
    ArrayList<Double[]> outputs;
    //Stores the layers of nodes in the network including hidden,input and output layers
    ArrayList<Node[]> nodeLayers;

    Network() {
        //initialises each array list
        nodeLayers = new ArrayList();
        outputs = new ArrayList<>();
    }
```

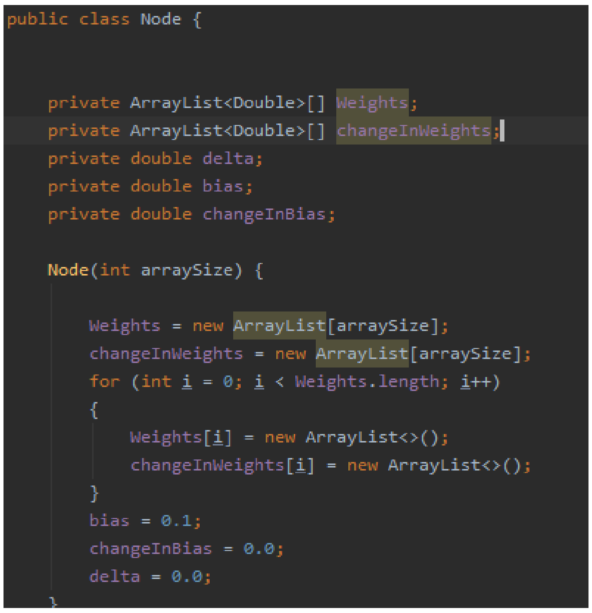
Figure 10 Network constructor

Node Class

A neural network is comprised of lots of nodes, which are formed to create a network. To represent this the node class is used. The function of this class is to be able to calculate all the relevant information for an induvial node. The includes calculating the nodes output, changing the weight attached to the node along with providing the relevant getters and setters. Each node object has the shared characteristics of containing weights, change in weights, delta and a bias. For the weights, they are stored as a 2-dimensional double

ArrayList. It is organised like this as makes processing the inputs from the individual items of clothing easier to manage. This is because the wardrobe object in the neuralNet class is also an 2 dimensional arraylist. This structure is mirrored so the weight of the clothes input node would correspond with the exact item of clothing within the wardrobe list. This also does not affect how any other nodes function as they will only have list of 1 dimension.

The nodes include change in weights which is based off the change in weights for that node's weight in the previous cycle. This is stored so that momentum can be included within the changes in weight. This is done as it can have the improve the speed and accuracy of the network along with having the affect to dampen the oscillation, making the resulting model more accurate. This is because having values go between positive and negative change, momentum with lessen the effects and push the change of weight more in the general trend in which the weights had been heading.

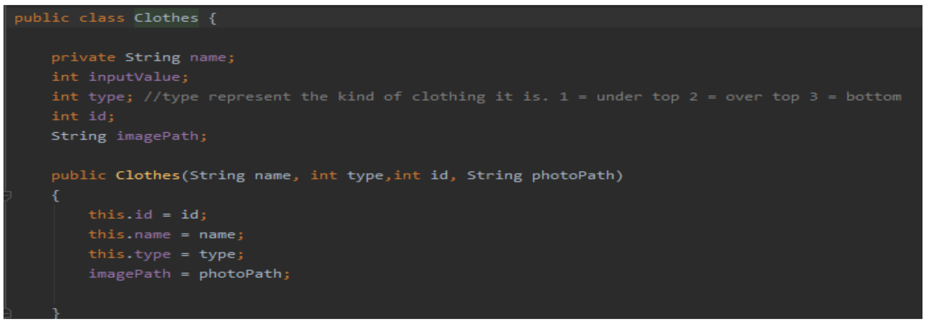


```
public class Node {


    private ArrayList<Double>[] Weights;
    private ArrayList<Double>[] changeInWeights;
    private double delta;
    private double bias;
    private double changeInBias;

    Node(int arraySize) {

        Weights = new ArrayList[arraySize];
        changeInWeights = new ArrayList[arraySize];
        for (int i = 0; i < Weights.length; i++)
        {
            Weights[i] = new ArrayList<>();
            changeInWeights[i] = new ArrayList<>();
        }
        bias = 0.1;
        changeInBias = 0.0;
        delta = 0.0;
```

Figure 11 Node Constructor

Clothes/Outfit Classes

This class is the basis for every item of clothing within the system. It achieves this by storing all the necessary information about an item being the name, type, id and image path. From this, a clothes object is defined. The expanded version of this class is Outfit with is an object comprised of 3 clothes objects which is this application definition of an outfit. Each of these are used to store info about the clothes and so are mostly used for getting and setting information about the items.



```
public class Clothes {

    private String name;
    int inputValue;
    int type; //type represent the kind of clothing it is. 1 = under top 2 = over top 3 = bottom
    int id;
    String imagePath;

    public Clothes(String name, int type,int id, String photoPath)
    {
        this.id = id;
        this.name = name;
        this.type = type;
        imagePath = photoPath;

    }
```

Figure 12 Clothes Constructor

4.2 - Main Activity Screen Implementation

The main activity's main function in to run the Outfit Recommender. This is run through the 3 buttons in the main screen. This allows for the loop the process loop of the user gets a recommendation, they decided if the like the suggested outfit or not, and the neural network is adjusted accordingly. To visualise the process, a space in the activity is dedicated to displaying the outfits. This is displayed with a purple outlined box in the centre of the screen. This contains a horizontal linear layout which contains 3 other vertical linear layout constraints. Within each 3 vertical linear layout constraints are a Textview object and a ImageView object. On resolution of the randomise button OnClick function it will set these objects to the relevant clothes as seen to the right.

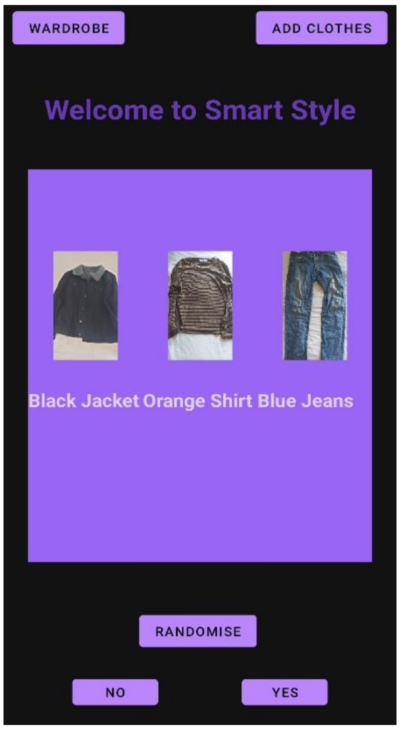
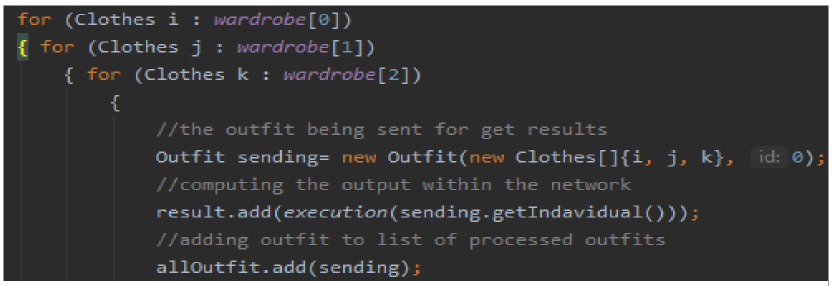


Figure 13 Main Screen with a recommender outfit

For the randomise button, its onClickListener triggers the function randomOutfit. This function tries to run the public function of a NeuralNet object called running and on resolution change set the textviews and imageViews in the screen to the chosen outfits. The NerualNet function running is what does the bulk of the processing.

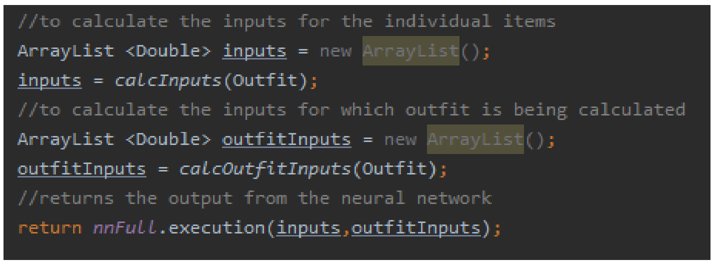
The running function's purpose is to do a full run of every possible outfit combination within the wardrobe stored within the NeuralNet object. Once it obtains every possible result, it then uses its selection function to decide which object gets shown. For calculating the outputs of each the outfit the function outfitProb is called. It is able to get every result by looping for each item in the wardrobe in each type. For this is creates an Outfit Object from the current cycles clothes objects, whose results are calculated and then stored in the function. This is so it can be later used for the selection process.



```
for (Clothes i : wardrobe[0])
{ for (Clothes j : wardrobe[1])
    { for (Clothes k : wardrobe[2])
        {
            //the outfit being sent for get results
            Outfit sending= new Outfit(new Clothes[]{i, j, k},  id: 0);
            //computing the output within the network
            result.add(execution(sending.getIndavidual()));
            //adding outfit to list of processed outfits
            allOutfit.add(sending);
```

Figure 14 Calculating all the outfit probabilities.

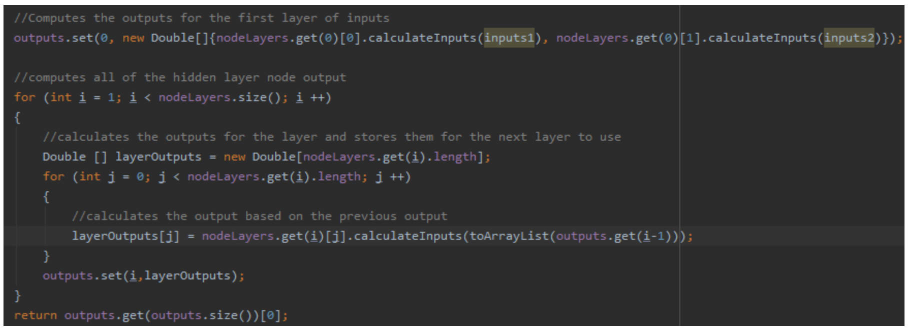
To calculate the output of a select outfit the execution function is used. This passes the outfit which is currently needing to be processed and running it through the neuralNet. It calculates the inputs which are needed for the neural network by sending the passe outfit into the calcInputs functions. These functions return back an ArrayList of 1's and 0's. These are the inputs for the neural network with the 1's representing the clothing item or outfit. This allows for only the potential outfit's item of clothing and the outfit itself can be considered when calculating the output.



```
//to calculate the inputs for the individual items
ArrayList <Double> inputs = new ArrayList();
inputs = calcInputs(Outfit);
//to calculate the inputs for which outfit is being calculated
ArrayList <Double> outfitInputs = new ArrayList();
outfitInputs = calcOutfitInputs(Outfit);
//returns the output from the neural network
return nnFull.execution(inputs,outfitInputs);
```

Figure 15 NeuralNet Execution function

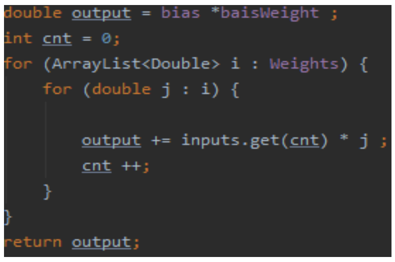
Once the inputs needed for the neural network have been calculated, they are passed into the Network object's function execution. In this function, it loops through every layer within the network. It initially calculates the outputs for the first layer based on the inputs of the clothes and the outfit. Form there is will loop for the amount of hidden layers which have been set up. The more layer which are implemented the better the performance in learning trends should be. Once the final layer of the network has been reached, the result from the output node will be sent as the return value.



```java
//Computes the outputs for the first layer of inputs
outputs.set(0, new Double[]{nodeLayers.get(0)[0].calculateInputs(inputs1), nodeLayers.get(0)[1].calculateInputs(inputs2)});

//computes all of the hidden layer node output
for (int i = 1; i < nodeLayers.size(); i ++)
{
    //calculates the outputs for the layer and stores them for the next layer to use
    Double [] layerOutputs = new Double[nodeLayers.get(i).length];
    for (int j = 0; j < nodeLayers.get(i).length; j ++)
    {
        //calculates the output based on the previous output
        layerOutputs[j] = nodeLayers.get(i)[j].calculateInputs(toArrayList(outputs.get(i-1)));
    }
    outputs.set(i,layerOutputs);
}
return outputs.get(outputs.size())[0];
```

Figure 16 Processing all the outputs for each layer in the neural network

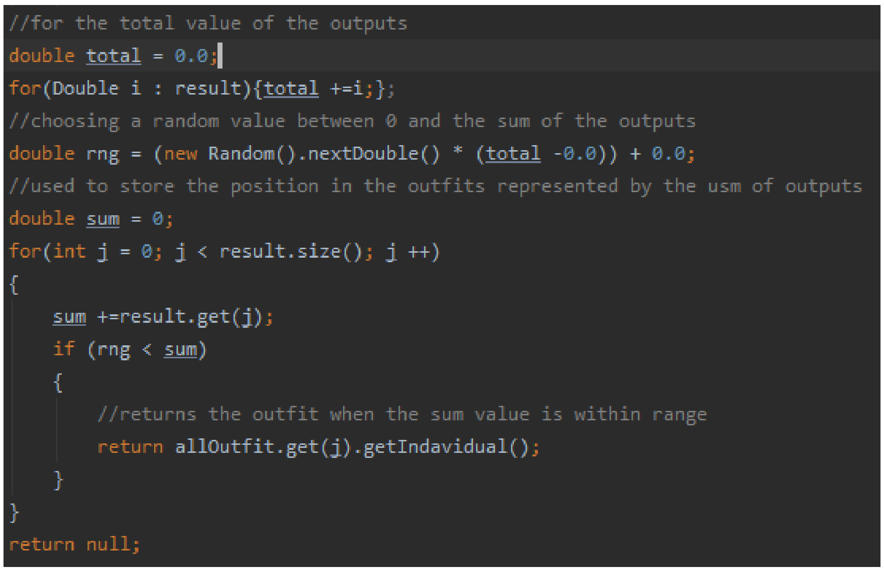
The output from each node in a layer is calculated from the weight attached to the node and the input which is passed into the function. It loops for every weight stored in the node along with the basis. The of this loop is return and stored for the next layer to use.



```java
double output = bias *baisWeight ;
int cnt = 0;
for (ArrayList<Double> i : Weights) {
    for (double j : i) {

        output += inputs.get(cnt) * j ;
        cnt ++;
    }
}
return output;
```

Figure 17 Calculating node Output.

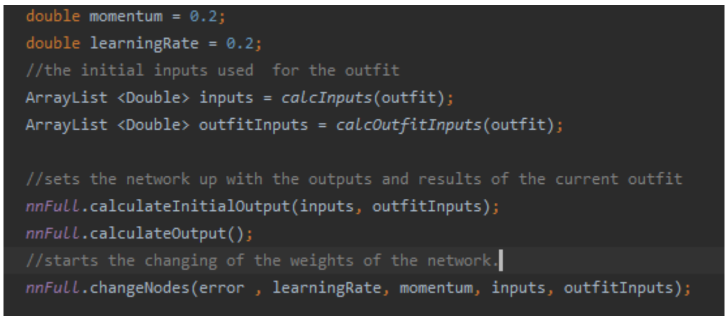
This process occurs for every outfit in the Object. Once all the outfits outputs have been calculated the selection algorithm is then used to choose which outfit is chosen. As the project is using a roulette selection method for the selecting the outfit, sum of all the outputs is found. This is used to choose a random double value between the sum and 0.0. The value produced is the value which will be selected as the outfit to be presented to the user. To obtain the index of the outfit, it loops though all the outfits, adding their respective results to the variable sum, which keep track of the sum of the outputs up to that point. It then checks to see if the new sum would be greater than the random position value and if so it will return that outfit. This returned outfit will be stored as the network's current best outfit. This done so that it can be displayed and also changed later when the network is learning.



```
//for the total value of the outputs
double total = 0.0;
for(Double i : result){total +=i;};
//choosing a random value between 0 and the sum of the outputs
double rng = (new Random().nextDouble() * (total -0.0)) + 0.0;
//used to store the position in the outfits represented by the usm of outputs
double sum = 0;
for(int j = 0; j < result.size(); j ++)
{
    sum +=result.get(j);
    if (rng < sum)
    {
        //returns the outfit when the sum value is within range
        return allOutfit.get(j).getIndavidual();
    }
}
return null;
```

Figure 18 Roulette selection method

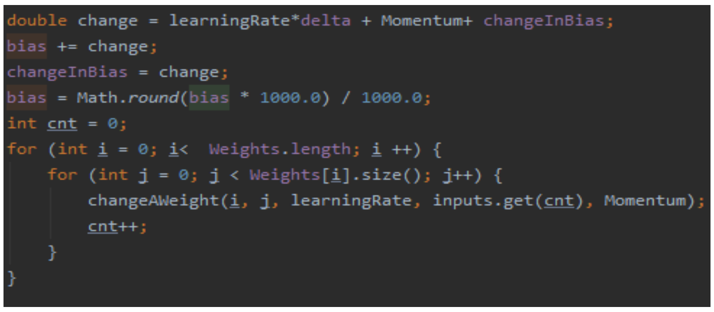
Once the result is shown the user can either like or dislike the outfit which they are presented with. Choosing either option will run the NeuralNet process outcomeChange. This function has the decision made by the user being representing liking the outfit being a 1 or a 0 respectively. From there the changeAllWeigths function is called while passing the decision and current best outfit. The reason why 1 or 0 is pass is that this allows for the calculation of the error to occur. This is because if the result is positive the error is calculated as the 1 – the output. this is because the best possible value and outfit will get is 1. This work the same way for if the decision is 0. In this function, the learning rate and the momentum for the network is set up. This will affect the rate in which the weight will be changed. From this the network is set up to the current best outfit. This restores all the output calculated in each layer of nodes. Once set up, the network will adjust the weights of the network based on the inputs and the initial layer calculated.



```
double momentum = 0.2;
double learningRate = 0.2;
//the initial inputs used  for the outfit
ArrayList <Double> inputs = calcInputs(outfit);
ArrayList <Double> outfitInputs = calcOutfitInputs(outfit);

//sets the network up with the outputs and results of the current outfit
nnFull.calculateInitialOutput(inputs, outfitInputs);
nnFull.calculateOutput();
//starts the changing of the weights of the network.
nnFull.changeNodes(error , learningRate, momentum, inputs, outfitInputs);
```

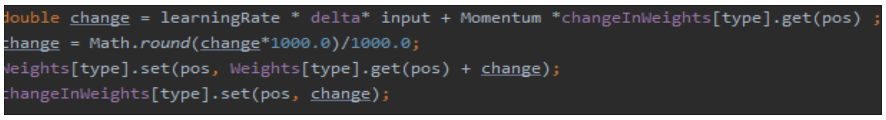
Figure 19 Configuring the weight learning system

Very similar to how the output was calculated, the weights are changed in a very similar manner. For all the nodes in the network their errors are set. This uses back propagation, so it works from the output layer and set the errors for each layer until the input layer. The error for each layer's node is the sum of all of the next Layer's node error multiplied by the connecting weight from the current layer's node and the next layer's node. This becomes each nodes delta and is used to change the weight in that node. Once the error has be set for every node in the layer, the weight for each respective node is then changed based on the set delta values and the output from the previous layers. To change all the weight in a node, it loops for every weight value in the node. For there each weight has it's individual weight changed based on the learning rate, momentum, delta and the input for that weight. The change in the weights is also always rounded to 5 decimal places.



```
double change = learningRate*delta + Momentum+ changeInBias;
bias += change;
changeInBias = change;
bias = Math.round(bias * 1000.0) / 1000.0;
int cnt = 0;
for (int i = 0; i<  Weights.length; i ++) {
    for (int j = 0; j < Weights[i].size(); j++) {
        changeAWeight(i, j, learningRate, inputs.get(cnt), Momentum);
        cnt++;
    }
}
```

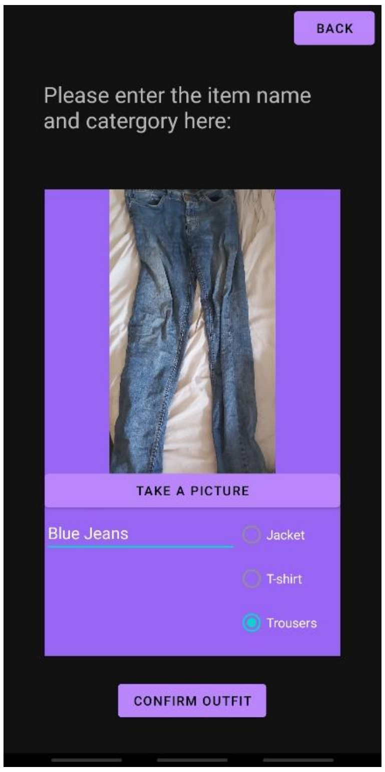
Figure 20 changing all the weights in a node.



```
ouble change = learningRate * delta* input + Momentum *changeInWeights[type].get(pos) ;
hange = Math.round(change*1000.0)/1000.0;
eights[type].set(pos, Weights[type].get(pos) + change);
hangeInWeights[type].set(pos, change);
```

Figure 21 Changing a singular weight.

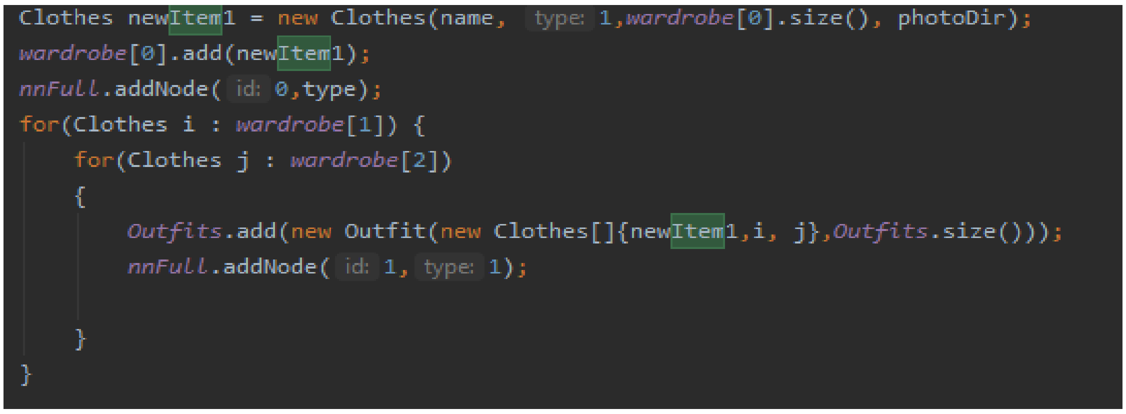
4.3 - Item Adding Screen Implementation



The main function of this screen is giving the user the ability to add items into the wardrobe. To do this the activity has been designed to clearly show what properties are needed to add an item to the wardrobe. This can be seen with the editable textView, the Radio buttons defining the categories and the button to take a picture of the item to be added. When the user wants to add an item by pressing the confirm item, it will try to add all the information from the activity and add them to a new Clothes Object. It will attempt all of these, but if any component is missing it will change the textview message at the top of the screen to tell the user what they need to fill in. This will prevent items being added which do not enough information while also informing the user on how to fix this.

With getting the type of clothing and the name of the item is done through pulling the set value through and passing them though. However, to get a picture taken another activity needs to be run. This intent is using a MediaStore class activity to start up image capturing. To be able to store the picture, which has been taken, a File object needs to be set up for the picture. The function will try to create a new File object and is it succeeds it will create a URI object. This will set up the location to where the File object will get saved to. On the resolution of the picture capturing activity the image is stored to where the URI is set up to. Along with this, the picture which was just taken would then be displayed on an imageView in the Item Adding activity. This is done by taking the file path where the image is stored and using the bitmap factory object to decode the file so that I can be displayed. Most of this implementation was inspired by the Android studio guide to implementing image capturing into applications. [23]

With all of the necessary information filled in for an item, the neural network will be able to execute the function, additem. This function uses the passed information to create a new clothes object which is then added to the wardrobe ArrayList at the respective type. As whenever an item is added to the wardrobe this creates new possible outfits. To add these new outfit, the new item is added with every other possible combination of the other 2 types. All of these outfit would then be added to the Outfits list. For the neural network, new weights need to be introduced. For the clothes inputs, a new weight is created for the respective type that the item is. For the outfit inputs, it will add a new weight to them for every new outfit which is created during this process.



```
Clothes newItem1 = new Clothes(name,  type: 1,wardrobe[0].size(), photoDir);
wardrobe[0].add(newItem1);
nnFull.addNode( id: 0,type);
for(Clothes i : wardrobe[1]) {
    for(Clothes j : wardrobe[2])
    {
        Outfits.add(new Outfit(new Clothes[]{newItem1,i, j},Outfits.size()));
        nnFull.addNode( id: 1, type: 1);

    }
}
```

Figure 22 An example of a type 1 item being added to the wardrobe.

4.4 - Wardrobe Activity Screen Implementation

This activity lets the user see all the item which are in the wardrobe and gives them the option to edit the items. When the user enters the screen, all the items are displayed in the purple wardrobe box. On this initial screen the user is met by a drop drown bar. This bar contains all 3 types which the clothes can be. The user can select a type from the spinner which on selecting the type which they want will only display items from that type. This is obtained from a Neural Net function called getAllNamesOfType. From the passed value, sent on the selection of the spinner type, it will loop for the length of that wardrobe type, adding each item's name into a string. This string is comma separated so that once the function has completed, the returned string then be separated and used for the TextView text.

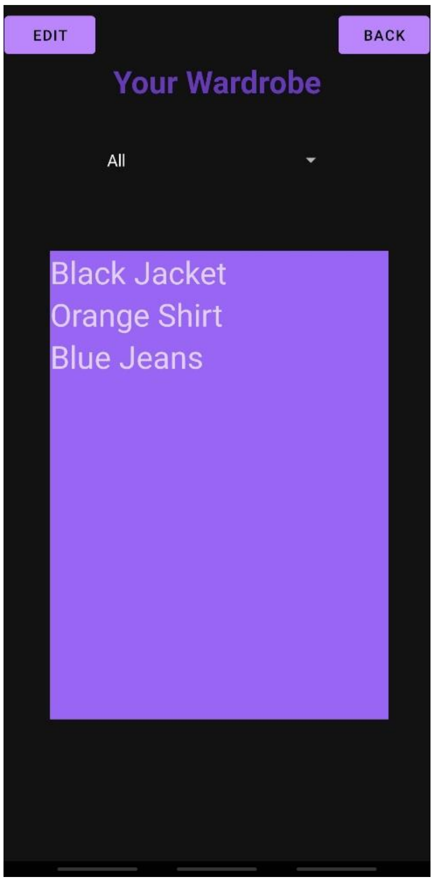
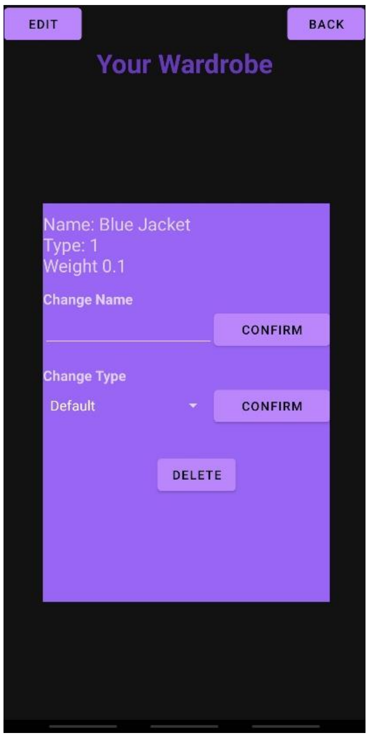


Figure 23 Displaying the wardrobe

In the wardrobe box, there a multiple textview object all stored in a linear layout constrictor. Whenever parts of the wardrobe are shown, the layout is looped for the number of children it has, being the textviews, and changing the text to the respective item of clothing.

When items are in the wardrobe box they can be selected. This is when a red box is draw around the item which is selected. This allows for the user to see which item they have have selected and tells the activity which item to edit. This is done through having each Textview object have an onClickListener. This is initialised once the Activity has been started and when the wardrobe is being read. This means whenever a TextView is pressed it will run the highlight function. This function works by having each TextView pass their ID within the linear layer constraint. This is then stored within the activity as it will be used for other functions. The function works where is no other object in the item has already been selected it will highlight the object which was passed. If there was already an item which was selected, it would turn that item’s background to transparent before changing the currently pressed object to the red highlighted colour. This works the same way if the an already highlighted item has been pressed again. Along with that it would also set the currently highlighted value to -1 to show nothing was selected anymore. This important as to make the editing screen work as intended, it needs to prevent the user form trying to edit no items due to no objects being selected. Along with that -1 is also a value which could not be produced by an index.

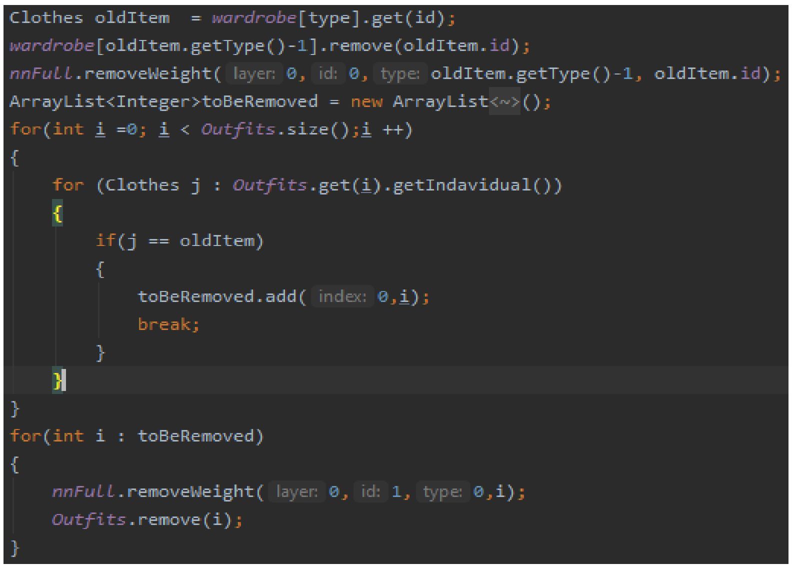
With an item selected the user would be able to select the edit button. This changes the wardrobe box to have all the information about the selected item. This is done by making the wardrobe layout constraints visibility set to gone and turning the editing layout constraints visibility to visible. In the Editing layout constraint, it shows the item’s name, Type, and weight in the neural network. Along with this they are presented with multiple options to change elements about the item they have selected. First is changing the name of the item, which reads the text int the editTextView and passes that to a function within the NeuralNet which uses a Clothes object function to set the new name.



Name: Blue Jacket
Type: 1
Weight 0.1

Figure 24 Editing screen for an item

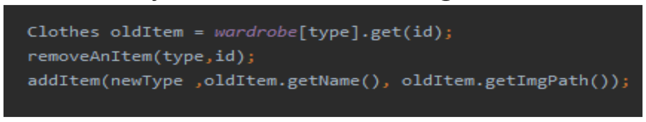
When the user wants to get rid of an item them can use the delete button which is provided. Certain considerations needed to be made as the item needs to be removed without changing the Neural network irreversibly. To do this the NeuralNet function removeAnItem is used. This function gets the desired Clothes object to be removed and stores it in a temporary clothes object. It first removes the item from the wardrobe itself using the old item’s id and type to remove the item at the correct location. After that the weight associated in the individual item’s inputs in the first layer of the network. Finally, it has to remove all the possible outfits it was a part off. To do this is loops for the entire Oufits list and adds the Outfits IDs when the item is found comprising the outfit. Once all the Outfits which need to be removed have been found, it will loop through the list from back the highest id to the last removing the Outfits objects and the associated Weights in the input layer one by one. It is done this was as when an item in the list is removed, any item items after it on the list are moved down the list meaning that their index would change. Therefor, it is needed to go from the newest outfit with the item in to the oldest. Once completed the selected object would have been removed from the Network and the user would be taken back to the original wardrobe screen.



```
Clothes oldItem  = wardrobe[type].get(id);
wardrobe[oldItem.getType()-1].remove(oldItem.id);
nnFull.removeWeight( layer: 0, id: 0, type: oldItem.getType()-1, oldItem.id);
ArrayList<Integer>toBeRemoved = new ArrayList<~>();
for(int i =0; i < Outfits.size();i ++)
{
    for (Clothes j : Outfits.get(i).getIndavidual())
    {
        if(j == oldItem)
        {
            toBeRemoved.add( index: 0,i);
            break;
        }
    }
}
for(int i : toBeRemoved)
{
    nnFull.removeWeight( layer: 0, id: 1, type: 0,i);
    Outfits.remove(i);
}
```

Figure 25 Process to delete an item

Another button is the changing type button. This changes the type of the item to any of the other types. This is done by taking the value selected by the Spinner object and pass it through to the NeuralNet. From there the setType function takes the all the information about the object and stores it. From there it deletes the current object form the wardrobe. This takes advantage of the preestablished removeAnItem function. Once removed from the wardrobe, a new item is introduced with all the old information from the item but with the new type. This will lead to the item being successful being transferred from one type to another without leaving the original behind. This ensures that the neural network doesn’t try and recommended the old object as it would no longer exist in the same capacity.



```
Clothes oldItem = wardrobe[type].get(id);
removeAnItem(type,id);
addItem(newType ,oldItem.getName(), oldItem.getImgPath());
```

Figure 26 Transferring process

Chapter 5. Testing and Discussion

5.1 - Test Description

The main aspect of the application which needs to be tested, is its performance in being able to learn the users taste in clothes. To test the network, the accuracy of the prediction after a set amount of train at multiple intervals would be used. This would test the network after training from 0 – 200 epochs going in iteration of 10 testing epochs to 100 where the next testing values would be 150 epoch and 200 epochs. For testing, a wardrobe and tastes for that wardrobe needed to be created. This was done by using a testProfile. The test profile would simulate a user using the app. With each profile it would contain all the items of clothing they have and the relationship ships between each item. This would act as their tastes. To decide whether or not the simulated user will like or dislike the outfit, a score it created from the outfit and the outfit score needs to be able to exceed the threshold. This threshold is a score of 1 or more compatible items of clothing. For example, profile A might have Top X which they like with Bottom Y but dislike with UnderTop Z. This will mean that the user will not choose the item as this outfit would produce a score of 0 as the relationship with Top X and Bottom Y would be +1 while the relationship between Top X and UnderTop Z would be -1. This would lead to an overall score of 0 which would not be accepted for the threshold.

This testing method does have limitations, however. This is due to not have having historical data of real people outfits choices. This means that it has to rely on a simulation of it which has the limitation of not be able to properly predict the how the user will react due to potential spontaneity of results. As well as this, as users taste change over time it means that the relationships between item would not be static and would be prone to changing. As well of this there are external factor when address what outfit to wear as the user may like the outfit but not be suitable for the current weather conditions leading to a favour outfit being rejected. Another limitation of testing the network is that there are elements of randomness, even though it is skewed. This means that multiple answer will be obtain from the same settings. To alleviate this for each test case, an average of 5 separate testing runs at the same setting will be used as the value for the respective epoch. This is done in the effort to get a more accurate results and being less susceptible to outliers in results. Along with this, it will be tested while have a single hidden layer of neurons.

The goal of the testing is to prove that the neural network can learn tastes over multiple recommendations. It should show an upward trend with the % of good recommendation increasing at a reasonable rate. It should also show the ideal settings of the neural network to be set at. This would be found by a combination of speed of learning, accuracy, and variance in results. The setting which will be changed are the learning rate, momentum, and the profiles. The hypothesis is that by increasing the learning rate the accuracy will increase at a faster rate, but may be prone to large variance due to potentially unlearning key bits of information, while lowering the rate should produces a slower but smoother learning curve. With the momentum, it's expected that it will help the learning when increased as it should be able to emphasise the direction the weights were learning in. This should help lessen the effects of a popular item being used in clashing outfit.

5.2 - Relationship tables

Profile 1

	Hoodie	Black Jacket	Band Shirt	Black Shirt	Grey Shirt	Black Jeans	Grey Jeans
Hoodie	N/A	N/A	+1	-1	0	0	+1
Black Jacket	N/A	N/A	0	+1	0	+1	0
Band Shirt	+1	0	N/A	N/A	N/A	-1	0
Black Shirt	-1	+1	N/A	N/A	N/A	-1	-1
Grey Shirt	0	0	N/A	N/A	N/A	+1	0
Black Jeans	0	+1	-1	-1	+1	N/A	N/A
Grey Jeans	+1	0	0	-1	0	N/A	N/A

Profile 2

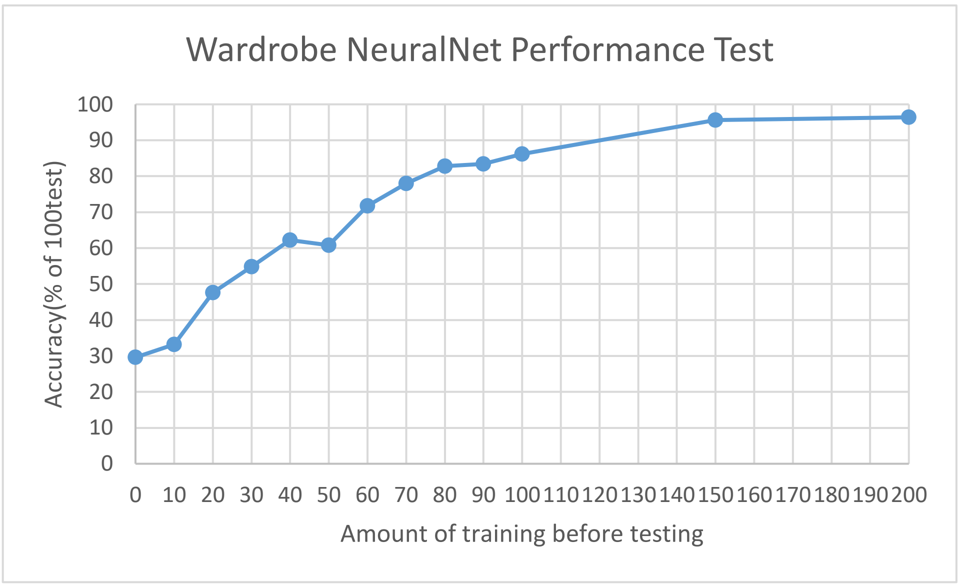
	Suit Jacket	Hoodie	Coat	Jumper	T-shirt	Long Sleave	Vest	Polo	Band Shirt	Smart Trousers	Trousers	Shorts
Suit Jacket	N/A	N/A	N/A	N/A	-1	0	-1	1	-1	+1	0	-1
Hoodie	N/A	N/A	N/A	N/A	+1	0	0	-1	+1	0	0	0
Coat	N/A	N/A	N/A	N/A	0	0	-1	0	0	+1	+1	-1
Jumper	N/A	N/A	N/A	N/A	0	0	0	0	0	0	+1	-1
T-shirt	-1	+1	0	0	N/A	N/A	N/A	N/A	N/A	0	0	0
Long sleave	0	0	0	0	N/A	N/A	N/A	N/A	N/A	0	+1	0
Vest	-1	0	-1	0	N/A	N/A	N/A	N/A	N/A	-1	0	+1
Polo	+1	-1	0	0	N/A	N/A	N/A	N/A		+1	0	0
Band Shirt	-1	+1	0	0	N/A	N/A	N/A	N/A	N/A	-1	0	0

Smart Trousers	+1	0	+1	0	0	0	-1	+1	-1	N/A	N/A	N/A
Trousers	0	0	+1	+1	0	+1	0	0	0	N/A	N/A	N/A
Shorts	-1	0	-1	-1	0	0	+1	0	0	N/A	N/A	N/A

These tables show all the relationship between all the items in the profile's wardrobe. Any item with the relationship value N/A means that either it's a duplicate item or that it is an item which cannot be in an outfit together and therefore can have no relationship.

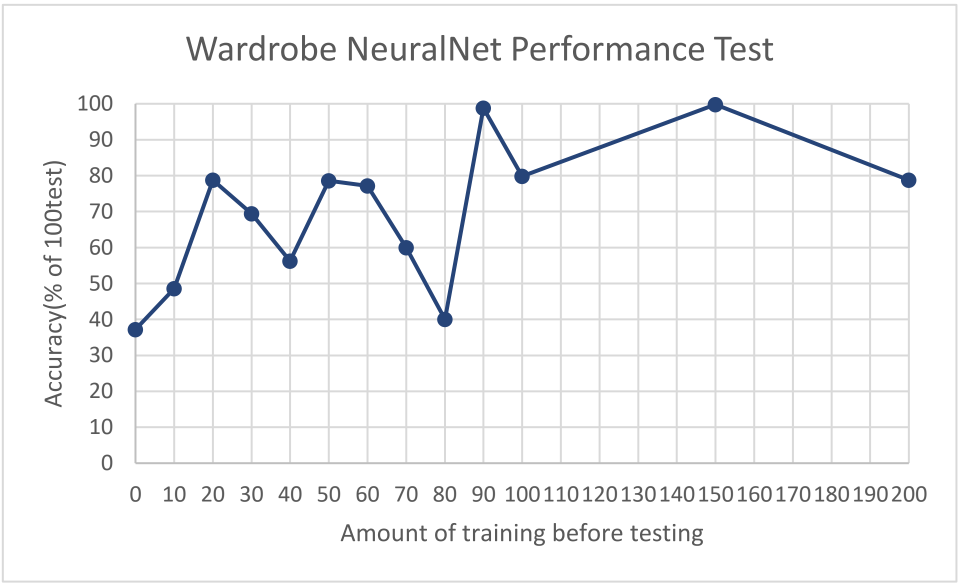
5. 3 - Test Cases

Test 1 – Profile 1, Learning rate = 0.2, Momentum = 0.2



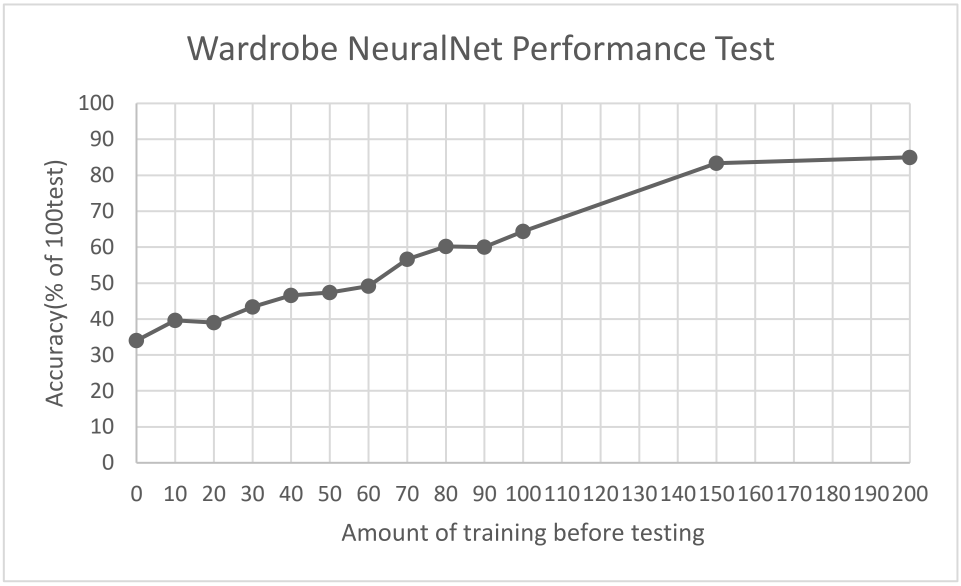
This test was to see if the NeuralNet would be able to learn a small sample size with the default settings. From the results in the graph, it shows that the NueralNet is capable to learn the users taste. This is displaced as there the trend in the result is generally positive. This shows that the more time the network is trained the great the accuracy normally is. Along with this it shows that the learning slows down the closer the network gets to getting 100% accuracy. This is because the increments in learning become much smaller the closer the accuracy gets.

Test 2 – Profile 1, Learning rate = 0.5, Momentum = 0.2



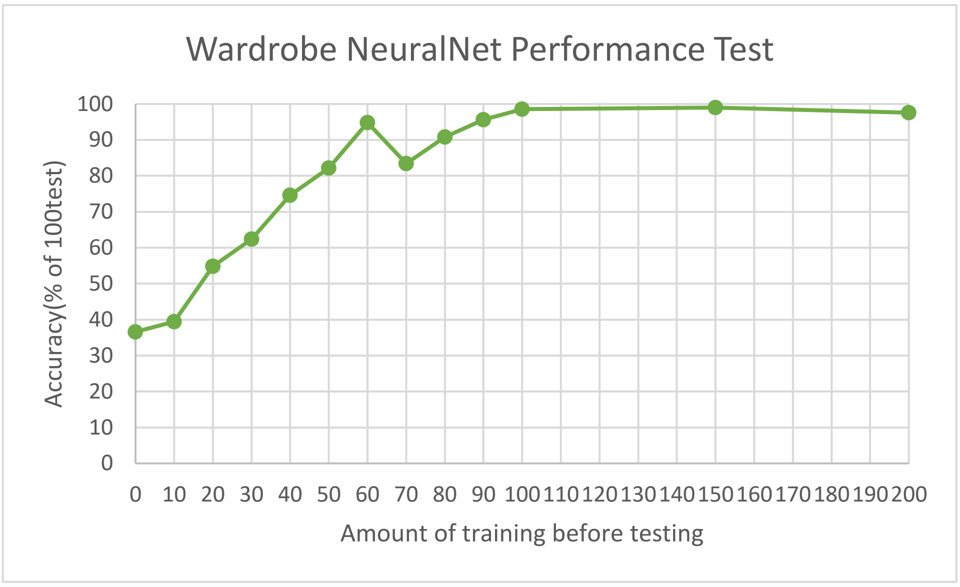
As expected by increasing the learning rate to over double, it has caused the consistency of the prediction to decrease significantly. This is seen as the graphs has severe jumps from both high and low accuracies. However, it does still follow a general upward trend showing that it is generally learning the problem. The peaks and valleys may be due to the fact that with such a significantly higher learning will cause the weights in the network to go through more substantial change per epoch. This could be leading to the network unlearning to much information on certain epochs causing a larger influence in the direction of the learning. This shows that plainly increasing the learning rate is not a good solution to learning the tastes faster as the loss of consistency is unacceptable.

Test 3 – Profile 1, Learning rate = 0.1, Momentum = 0.2



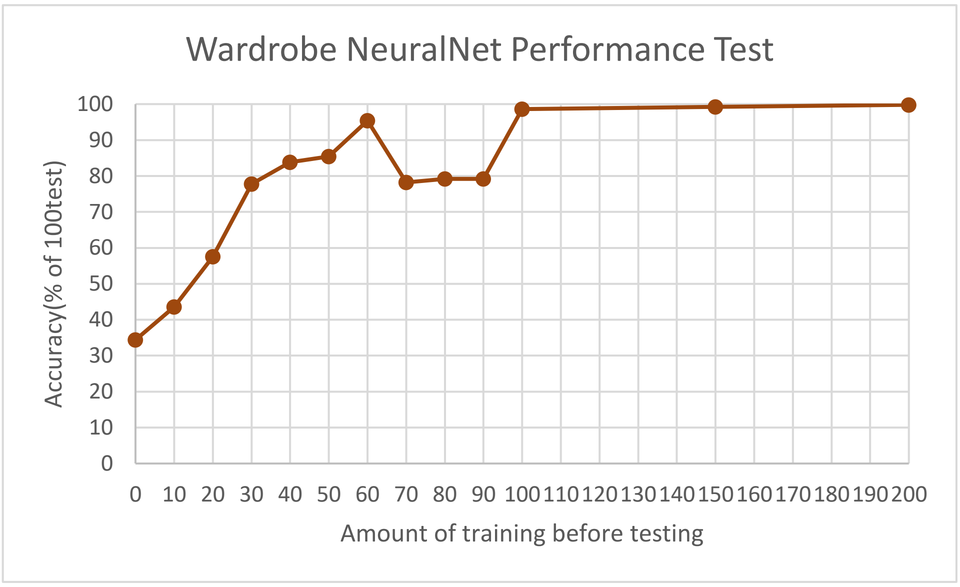
In this test it has shown a very smooth curve for learning. The curvature of the learning is fairly shallow and is close to being more a line. This is indicative of the more consistent nature of the network learning the information. As expected, this is in anthesis of increase the learning rate which produced more inconsistency. However, a side effect of lowering the learning rate is that the time to learn more of the taste has increased as the peak in this is fairly lower than pervious examples which were able to get near perfect at their peak. This shows that even though a lower learning rate does improve consistency, there needs to be a middle ground where it can learn faster without hurt the consistency. This is because the average user would be unlikely to reach usage in the hundreds meaning it is vital to be able to provide a good experience without it taking too long to learn.

Test 4 – Profile 1, Learning rate = 0.2, Momentum = 0.5



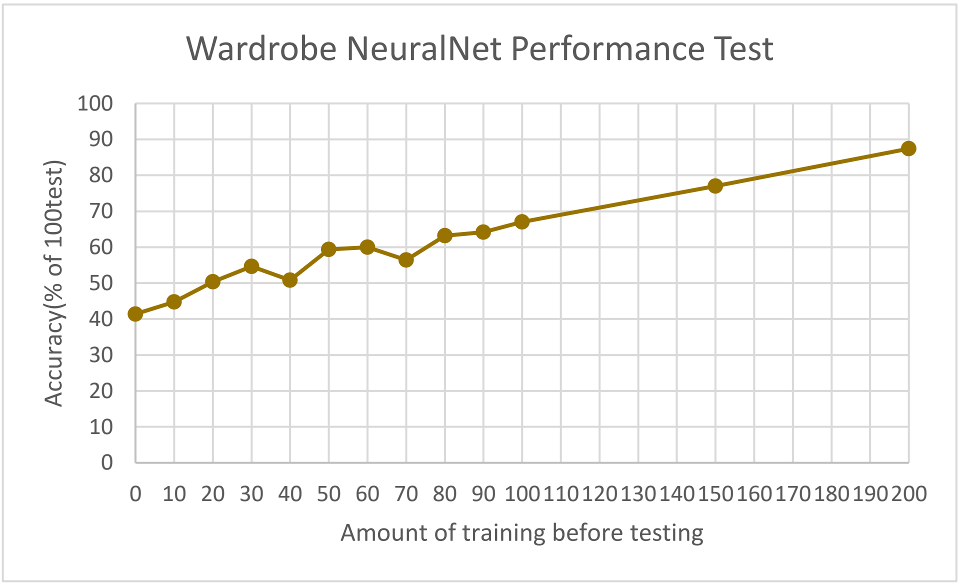
When compared to the initial settings used for testing, the increase in momentum follows a very similar upward trend in learning creating a mostly smooth curve with as few outliers. Unlike when the learning rate was increased, the learning is seen to be much consistent. This is while being at a faster rate of learning than the initial settings. This is possibly due to the how momentum only emphasises the trend the weight had previously been on rather than in the learning rate's case where it is control of the size of the adjustments made. From these results it shows that most likely, a higher momentum will be a greater benefit to improving the speed of learning without sacrificing the consistency.

Test 5 – Profile 1, Learning rate = 0.35, Momentum = 0.35



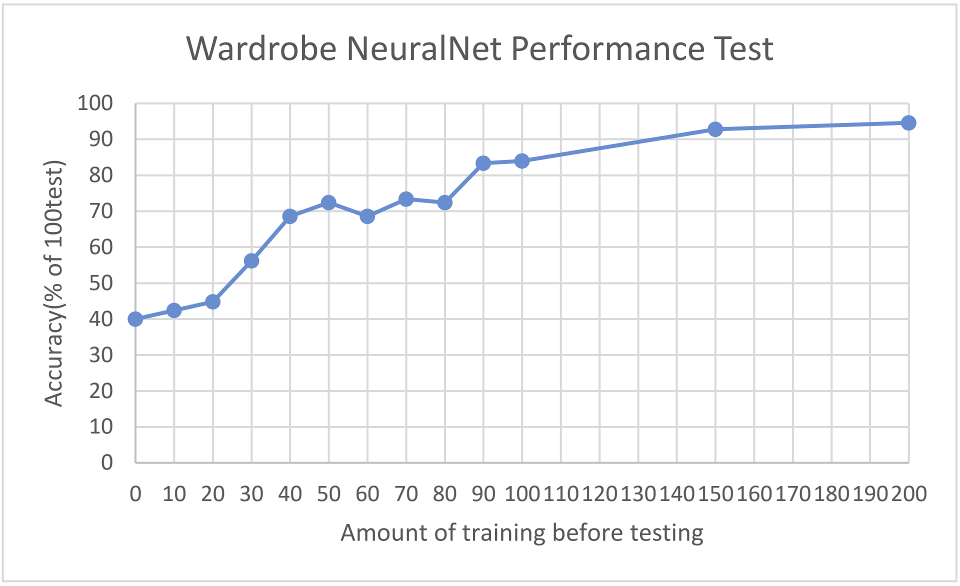
This graph shows the dramatic effect increasing both the momentum and the learning rate in tandem can have with the performance of the neural net. It shows how initially it learns the problem quickly as it takes around 30 epochs to start producing accuracy of >80%. This is overall an improvement as compared to the initial settings which took around 70 epochs of training to reach a similar value. This also produces a similar result as with only increasing the momentum as that reached this milestone at 50 epochs. This demonstrates that with the combination of both momentum and learning rate being increased the learning is improved considerably and more than each of them individually being altered. However, this also emphasises prior issues as there is an area of inconsistency introduced as from 70 to 90 epochs of training the overall performance dropped. This was most likely due to the influence of learning rate indicating that to obtain better results for the task as the learning rate being increased at a lower rate than the momentum may be beneficial.

Test 6 – Profile 2, Learning rate = 0.2, Momentum = 0.2



This graph displays how even when the problem is scaled up it is still able to solve the problem and produce results with high accuracy. Unlike the comparatively smaller wardrobe size of the first profile, the rate at which the accuracy improves has flattened out much more than the curve seen in the original. This highlights a potential issue with scalability as the larger the wardrobe the slower the learning may become if the settings are changed to alleviate this. Though the network work is still able to produce good accuracy despite this, just over a longer period of testing. This is due to the fact that the wardrobe size is 5 times larger as profile 1 has a total of 12 outfit whiles profile 2 has a total of 60 outfits making it theoretically 5 times harder to learn the tastes. To improve the performance with a larger wardrobe needs to be considered.

Test 7– Profile 2, Learning rate = 0.25, Momentum = 0.35

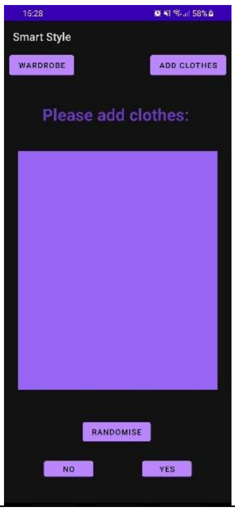
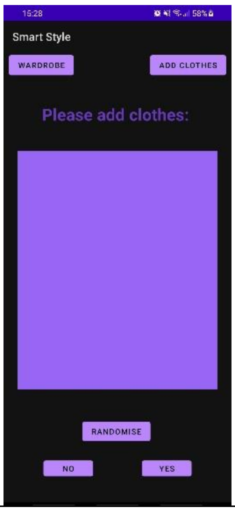


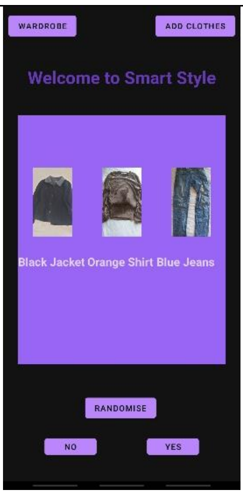
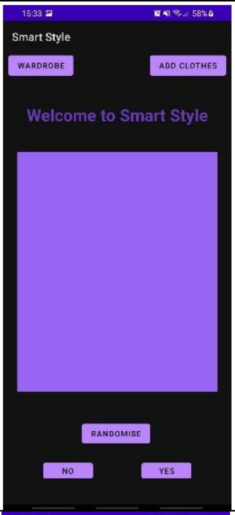
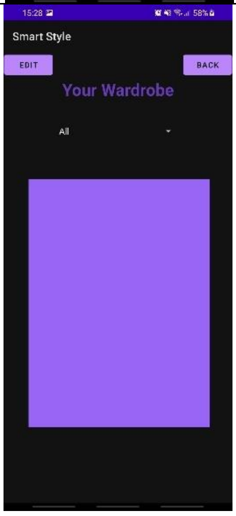
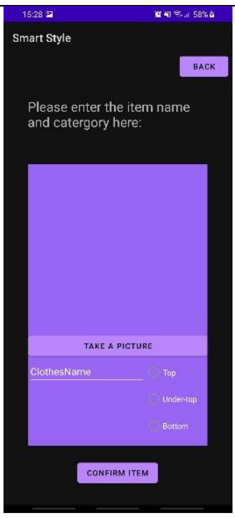
As tested previously, the setting of a higher momentum and a slightly high learning rate have proved to be successful in improving the performance of the network when given a larger amount of clothes to work with. This curve, while having dips, is generally smooth showing that progression in learning the problem is fair standard by having larger jumps in performance early and the difference in performance decreasing the longer more it's learned. One thing which is highlighted is that even with the performance being better, it never reaches the same peaks as with a smaller wardrobe. However, this is to be expected due to the increase in possibilities and relationships to learn. Though the peak performance seen here at over 90% would still be very satisfactory for the goal of the network.

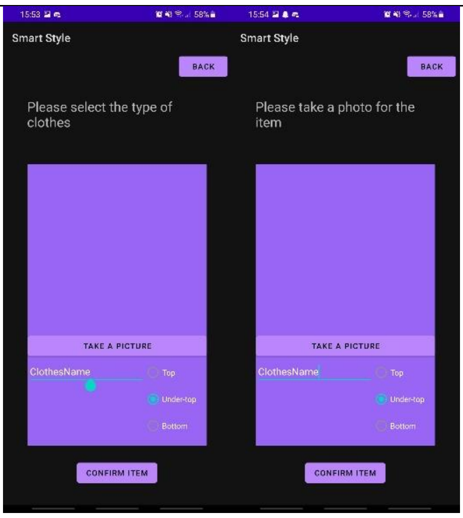
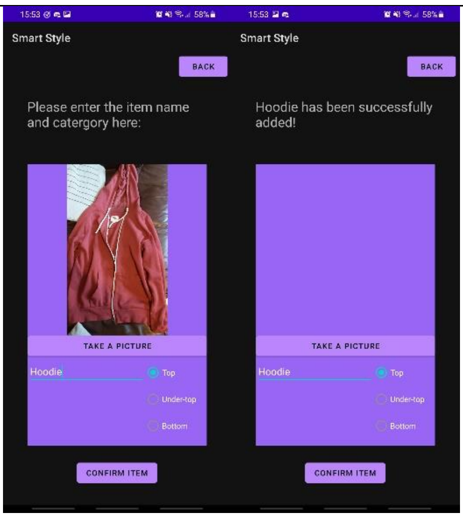
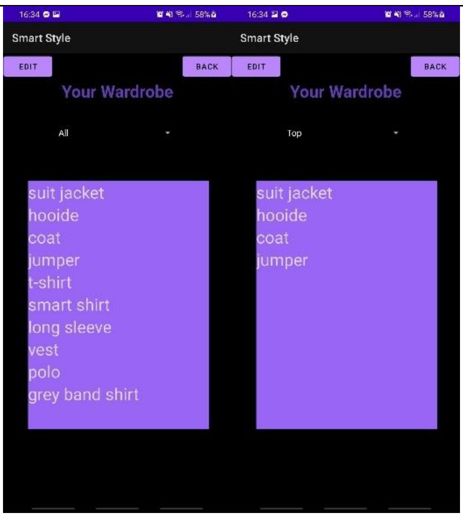
Overall, the testing has displayed that the neural is very much capable of learning the relationships between item of clothing in the wardrobe and is able to learn to pick better outfits over times. It has also been discovered that in long term usage of the application there may arise some scalability as the network may take a lot longer to learn tastes with greater scale of outfits. This could be solved in the future by implementing changing learning rate/Momentum depending of the size of the wardrobe. As well it's has be discovered that momentum is the more consistent tool to speed up the learning of the network. This is due to how is keeps the weight on its original trend meaning bad results will affect it less if it was previously highly favoured. Along with this learning rate was found to be good at increasing the peaks at which the network would learn but would also be very volatile as the larger the jumps in every time the network learns the bigger the impact of the future of the network is which can lead to the wardrobe tastes being unlearned, hurting the performance and the consistency.

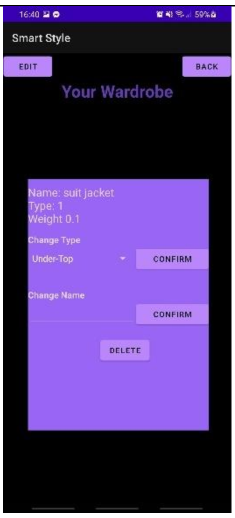
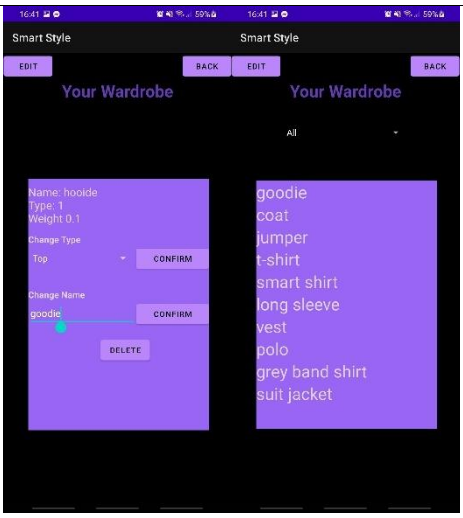
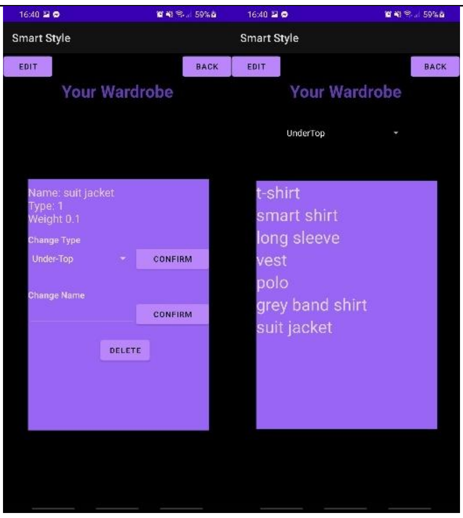
5.4 – Validation and Verification Testing

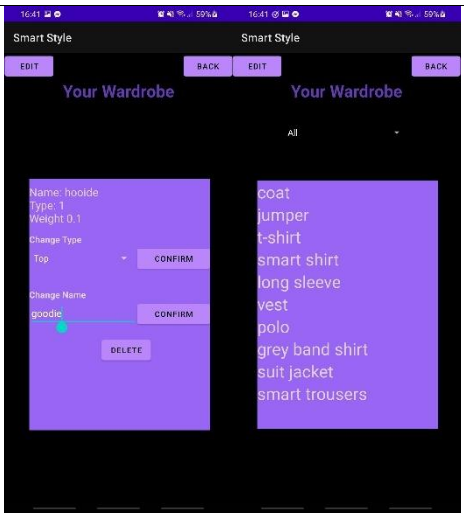
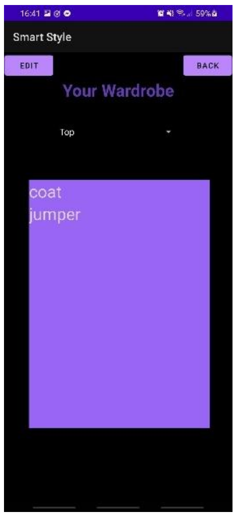
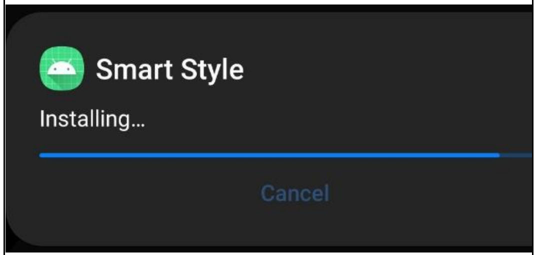
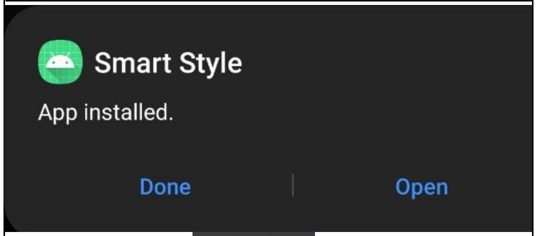
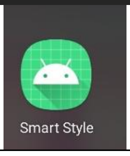
These tests are done to ensure that only the correct information can be input into the application and to see how the app reacts to incorrect data being input. It also intended to ensure that each of the function is working as expected.

Test Cases	Expectation	Result	Evidence
Recommendation button is pressed with no wardrobe initialised	No outfit will be recommended, and a message will be displayed	A message is displayed telling the user to add clothes	 15:28 Smart Style WARDROBE ADD CLOTHES Please add clothes: RANDOMISE NO YES
Recommendation is pressed with a wardrobe but no outfits	A message will be displayed and not recommendation is made	A message is displayed to add more clothes	 15:28 Smart Style WARDROBE ADD CLOTHES Please add clothes: RANDOMISE NO YES

Recommendation button is pressed with a outfit	A recommendation will be made, and the names and images of the clothes are shown	A recommendation clothes are shown on the screen	 WARDROBE ADD CLOTHES Welcome to Smart Style Black Jacket Orange Shirt Blue Jeans RANDOMISE NO YES
Decisions buttons (Yes/No) are pressed with no recommendation	A message is displayed, and nothing adjusting happens	No recommendation is displayed but no message is shown	 15:33 58% Smart Style WARDROBE ADD CLOTHES Welcome to Smart Style RANDOMISE NO YES
Menu (Wardrobe / Add Clothes) buttons functionality	The respective screen will be loaded.	The correct screen was loaded for each button	  15:28 58% Smart Style BACK Please enter the item name and catergory here: TAKE A PICTURE ClothesName Top Under-top Bottom CONFIRM ITEM

Can the user add an item of clothing which does not have all the relevant information	The item will be added to the wardrobe	The item was added with all the information provided and a message saying it's occurred	 Smart Style BACK TAKE A PICTURE ClothesName Top Under-top Bottom CONFIRM ITEM Smart Style BACK TAKE A PICTURE ClothesName Top Under-top Bottom CONFIRM ITEM Please select the type of clothes Please take a photo for the item
Does an item with full information get added to the wardrobe?	The item is added to the wardrobe with a message displaying this.	The item of clothing was added with a message. It could also been seen when editing the wardrobe	 Smart Style BACK TAKE A PICTURE Hoodie Top Under-top Bottom CONFIRM ITEM Smart Style BACK TAKE A PICTURE Hoodie Top Under-top Bottom CONFIRM ITEM Please enter the item name and catergory here: Hoodie has been successfully added!
Can item types be shown individually in the editor?	It would list all the item form that respective type	All the clothes are displayed in a list	 Smart Style EDIT BACK Your Wardrobe All Smart Style EDIT BACK Your Wardrobe Top suit jacket hooide coat jumper t-shirt smart shirt long sleeve vest polo grey band shirt suit jacket hooide coat jumper

Can an item's information be reached?	The information of the item should be shown with info like names.	The items information was displayed with all the options	 Smart Style EDIT BACK Your Wardrobe Change Type CONFIRM Change Name CONFIRM DELETE Name: suit jacket Type: 1 Weight 0.1 Under-Top
Can an item's name be changed?	The name of the item should become whatever the use has input	The name of the item has been updated to contain the new name of the item	 Smart Style EDIT BACK Your Wardrobe Change Type CONFIRM Change Name CONFIRM DELETE All Name: hooide Type: 1 Weight 0.1 Top goodie goodie coat jumper t-shirt smart shirt long sleeve vest polo grey band shirt suit jacket
Can an item's type be changed?	The item should move to the type which has been designated	The item has moved from one type to another.	 Smart Style EDIT BACK Your Wardrobe Change Type CONFIRM Change Name CONFIRM DELETE UnderTop Name: suit jacket Type: 1 Weight 0.1 Under-Top t-shirt smart shirt long sleeve vest polo grey band shirt suit jacket

Can an item be deleted?	The item should no longer appear in the wardrobe	The item no longer can be found in the wardrobe and	 Smart Style Your Wardrobe Name: hooide Type: 1 Weight 0.1 Change Type: Top Change Name: goodie EDIT BACK CONFIRM DELETE All coat jumper t-shirt smart shirt long sleeve vest polo grey band shirt suit jacket smart trousers  Smart Style Your Wardrobe EDIT BACK Top coat jumper
Can the app be installed on another device?	The app should install onto the android system	The app was able to be installed and could be run	 Smart Style Installing... Cancel  Smart Style App installed. Done Open  Smart Style

Chapter 6. Conclusion

In conclusion, the project has been able to produce an android application for purpose of outfit recommendation. It has been able to implement a neural network into the app which is designed to learn the tastes in each user's clothes. The neural net was able to implement Deep learning-esque model implemented a multi-layer perceptron network to better learn the information providers and learn the trend of information. The network has also been proven that through usage over time, the accuracy of the it increases to a point where most of the predictions are ones which would be received positively. This shows that it can learn the tastes within a reasonable time improving the user experience through more accurate recommendation. Along with this by having the network be proven to give improved prediction over time, it will be able to reduce the amount of decision which have to be made by the user, which will have fulfilled the goal of reducing the amount decision fatigue over time.

The functionality of the app has also been successfully implemented as it has multiple screens developed for multiple specific function. It has been able to give the user the ability to receive recommendations on outfits, add new items of clothing into the app, and to view, edit, and delete any item within the app. It was also able to produce a visual representation of each outfit being recommend from the picture taken of each object being displayed on the recommender screen activity. The editor screen has also given the user all the functionality to change items in the wardrobe on a whim, fulfilling the need aim of creating customisability in the application.

Overall the project has been able to satisfy all the basic aim which it set out to achieve with a functional android application being the end result of the development.

Future Works

The project has been able to succeed in creating the foundation for the desired application. However what has been created is still very surface level. Even though it was able to implemented a Neural Network, this network runs on a few layers of hidden layers. This leaves room for this to be expanded for greater learning. Along with this the way that each item of clothing is ranked is simplistic as it only considers the item as an individual item and as part of an outfit. This is compounded by the limited but broad classification system for the items of clothing. This leading to limiting the design of outfits. As well even though the feature available with this app are satisfactory more could be implemented to expand to greatly improve the level customisability.

In the future if the project was to be expanded, I would have expanded the depth to the functionality of the application. This would be seen mostly in the amount of information each item of clothing had. One way this could be expanded in the future would be having temperature values associated with each item. This would have allowed features such as temperature of the outfits being a consideration in the selection process. This would have been done by taking the current weather at the user's location and using that to filter out outfits which would exceed the range of outfits which would be comfortable in that weather. Along with using weather as another factor in recommendation, the clothes classification system would be expanded to the have more sub categories. This would not only allow for more specific descriptions of each item of clothing, it would allows for each subtype of clothing to having their own influence in the recommendation process. This would mean that each item would be judged by each item's individual score, the outfit score but also the score produced by the sub-type of clothes. This would provide more data to for the network classify the items with which would potentially lead to even better accuracy.

To build of what has already been created, implementation of social media aspect could be a good way to expand the current system. Implementing this could increase the user's satisfaction due to being able to connect and show of their outfit they got recommended to them with other users along with being able to get inspiration from other users. As well as this by creating implementing more online aspects to the app, it could use more use collaborative filtering technique to speed up the initial learning of the users taste by basing them of similar taste of other users in the application.

Chapter 7. References

[1]. Rykrsmith, 2012)

Rykrsmith, E. (2012). Dress Code or Not, What You Wear Matters. Quickbase. Retrieved 28 April 2021, from https://www.quickbase.com/blog/dress-code-or-not-what-you-wear-matters.

[2]. ("Amazon's Recommendation Engine: The Secret To Selling More Online", 2019)

Amazon's Recommendation Engine: The Secret To Selling More Online. Rejoiner.com. (2019). Retrieved 28 April 2021, from https://rejoiner.com/resources/amazon-recommendations-secret-selling-online/.

[3]. ("Nervous system - The nerve cell"

Nervous system - The nerve cell. Encyclopedia Britannica. Retrieved 22 April 2021, from https://www.britannica.com/science/nervous-system/The-nerve-cell

[4]. (Chandra, 2017)

Chandra, A. (2017). McCulloch-Pitts Neuron — Mankind’s First Mathematical Model Of A Biological Neuron. Medium. Retrieved 22 April 2021, from https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1.

[5]. ROSENBLATT, 1958)

ROSENBLATT, F. (1958). THE PERCEPTRON: A PROBABILISTIC MODEL FORINFORMATION STORAGE AND ORGANIZATIONIN THE BRAIN [Ebook]. Retrieved 21 April 2021, from https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf.

[6]. ("What is Deep Learning?", 2020)

What is Deep Learning?. Ibm.com. (2020). Retrieved 21 April 2021, from https://www.ibm.com/cloud/learn/deep-learning.

[7]. (Team, 2017)

Team, S. (2017). Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant. Apple Machine Learning Research. Retrieved 22 April 2021, from https://machinelearning.apple.com/research/hey-siri.

[8]. ("Welcome - Faceswap", 2021)

]Welcome - Faceswap. Faceswap. (2021). Retrieved 22 April 2021, from https://faceswap.dev/#content.

[9]. (Ricci, 2011)

Ricci, F. (2011). Introduction to Recommender Systems. Retrieved 21 April 2021, from.

[10]. (Ilic & Kabiljo, 2015)

Ilic, A., & Kabiljo, M. (2015). Recommending items to more than a billion people - Facebook Engineering. Facebook Engineering. Retrieved 22 April 2021, from https://engineering.fb.com/2015/06/02/core-data/recommending-items-to-more-than-a-billion-people/.

[11]. (Kuraa, 2019)

Kurama, V. (2019). A Simple Introduction to Collaborative Filtering. Built In. Retrieved 22 April 2021, from https://builtin.com/data-science/collaborative-filtering-recommender-system.

[12]. (Lamothe, 2019)

Lamothe, C. (2019). Decision Fatigue: What It Is and How to Avoid It. Healthline. Retrieved 21 April 2021, from https://www.healthline.com/health/decision-fatigue.

[13]. (Sharma, 2019)

Sharma, A. (2019). Neural Collaborative Filtering. Medium. Retrieved 22 April 2021, from https://towardsdatascience.com/neural-collaborative-filtering-96cef1009401.

[14]. Covington, P. Deep Neural Networks for YouTube Recommendations. Storage.googleapis.com. Retrieved 22 April 2021, from https://storage.googleapis.com/pub-tools-public-publication-data/pdf/45530.pdf.

[15]. ("combyne", 2021)

combyne. Combyne.com. (2021). Retrieved 28 April 2021, from https://www.combyne.com.

[16]. ("Shuffle Outfits", 2021)

Shuffle Outfits. Play.google.com. (2021). Retrieved 28 April 2021, from https://play.google.com/store/apps/details?id=net.robos.shuffleoutfit.

[17]. (Werbos, 1974)

Werbos, P. (1974). Beyond Regression: New Tools for Prediction and analysis in the behavioral sciences.

[18]. McGonagle, J. (2021). Backpropagation | Brilliant Math & Science Wiki. Brilliant.org. Retrieved 21 April 2021, from https://brilliant.org/wiki/backpropagation/.

(McGonagle, 2021)

[19]. Divakar, V. (2019). Understanding The Chain Rule. QuantInsti. Retrieved 21 April 2021, from https://blog.quantinsti.com/understanding-chain-rule/.

(Divakar, 2019)

[20]. ("Find out why Tinder® is the world's best dating app | Tinder", 2021)

Find out why Tinder® is the world's best dating app | Tinder. Tinder. (2021). Retrieved 28 April 2021, from https://tinder.com/about-tinder.

[21]. ("101+ Android Stats You Should Know in 2021 [Infographic]", 2021)

101+ Android Stats You Should Know in 2021 [Infographic]. TechJury. (2021). Retrieved 28 April 2021, from https://techjury.net/blog/android-market-share.

[22]. ("Meet Android Studio | Android Developers", 2021)

Meet Android Studio | Android Developers. Android Developers. (2021). Retrieved 28 April 2021, from https://developer.android.com/studio/intro.

[23]. ("Take photos | Android Developers", 2021)

Take photos | Android Developers. Android Developers. (2021). Retrieved 28 April 2021, from https://developer.android.com/training/camera/photobasics.