澳 門 理 工 學 院
Instituto Politécnico de Macau
Macao Polytechnic Institute

School of Public Administration
Bachelor of Science in Computing

COMP491 Final Year Project
Final Report

Academic Year 2017/18

Analyzing Fashion Trends using Machine Learning

Project number:	30
Student ID:	P1408576
Student Name:	Julight Zhong, Jiachen
Supervisor:	Dr. Rita Tse
Assessor:	Mr. Zachary Chui

Submission Date:	April 24, 2018

Abstract

Fashion plays an important role in both personal daily life and the global market. Although the scale of fashion industry is huge, there are few studies focused on using artificial intelligence to predict fashion trends. This type of research is difficult to gain focus since the popularity of fashion is quite abstract. Fashion trends have unpredictable potential risks which are difficult to avoid in the business strategies and decisions of fashion companies.

However, the use of social networks combined with deep learning yields a good opportunity for this type of study. Social networks can provide huge amounts of information, while deep learning allows extracting and processing massive amounts of data. Therefore, the idea of this project focuses on using an advanced machine learning model, deep neural networks, to predict the quality of fashion trends by training the model with data collected from fashion-related social networks. Specifically, it aims to use fashion related data, images, and the popularity marks of fashion which are voted on by social network users. These data were used as training material for deep neural networks. The voted popularity mark is the ground truth label of the output of the neural networks. To use collected fashion images, two types of popular convolutional neural networks were employed in this project.

For this project, four plans were employed to achieve the goal of predicting the fashion mark. Among the four plans, three were separately trained models, meaning that the convolution layers were trained separately without fully connection layer. These type of plans perform relatively poorly but can be trained within a reasonable time, and does not need a very good training machine. The last plan trained the convolution layer with fully connection layer as a whole model which is common in modern deep learning research. This model had better performance than the other three, but required much greater computational resources and longer training time.

The difficulties in the project included: construction and training of neural networks, collection of a large amount of data and images, and processing management. As a result, the details of design, analysis, critical decision and implementation to solve these difficult problems are presented in this project report.

Acknowledgement

I’d like to express my sincere appreciation to all those who have helped in the preparation and writing this paper.

Firstly, I would like to take this opportunity to show my sincere gratitude to my supervisor, Dr. Rita Tse, who has offered strategic and technical guidance for this project.

Secondly, I would like to thank Professors Giovanni Pau and Gustavo Marfia for giving me important inspirations for the project design.

Additionally, I would like to thank Mr. Zachary Chui and Dr. Chan Tong Lam who have been the assessors for the report and presentation.

I would like to express my appreciation to Mr. Ka Hou Chan, Mr. Ka Seng Chou and Dr. Xu Yang for providing support on the training machines and GPUs.

Meanwhile, not only to the professors, tutors, assessors, and other people whom I am acquainted with, but many others whose names I do not know also deserve my deepest appreciation. Because of many open-source software and free applications which are produced by a myriad of contributors, this project was able to be launched, developed and successfully finished.

Last but not least, I would also like to express my most sincere gratitude to my family and friends for their support both financially and spiritually.

Table of Contents

Table of Figures

List of Tables

Introduction

Nowadays, fashion plays a very important role in people's daily life. Fashion is a huge industry and creates plenty of commercial opportunities as well as risks. One of the major problems for fashion or apparel companies is predicting popular fashion trends. This information is critical for business strategy decisions. For example, if a fashion company is able to know what is popular this year, the company will put more resources for the popular product and avoid wasting money on producing unpopular products. More importantly, if the company can forecast what will be popular next year, they may make manufacturing preparations for next year to win the business competition. The current fashion industry market is around three trillion dollars and accounts for about two percent of global Gross Domestic Product (GDP) [1], these kinds of technologies and analysis tools hold huge potential for commercial value.

Machine Learning is a subfield of computer science which gives “computers the ability to learn without being explicitly programmed.” [2] In some extent, these kinds of algorithms break the traditional way in which the machine, or computer are designed and process data by following several instructions. Generally, it is a kind of data-driven algorithms which may summarize with regularity from input data and make the prediction or decision based on the history. Many approaches were created with the development of this area of study such as decision trees [3], SVMs (support vector machines) [4] and neural networks [5] etc. These models have the ability to predict information based on the history of data, and thus can be used for fashion forecasting.

With practical advantages of fashion forecasting and the development of machine learning researches in recent years, works related to fashion forecasting using machine learning approaches will be discussed in detail in chapter two. Though some previous researches have been conducted, the number of these studies is limited. There are basically two reasons. The first is the limited amount of data, and the second is the complexity of fashion trends which made predictions difficult. For the first obstacle, since the machine learning is a data-driven approach, bad quality or lack of the data will obviously affect the performance of the model. And for the second, fashion is very subjective, and it varies from person to person, and the standards of fashion are hard to formulate. To explore and perform improvement in this area, solving these two problems are critical for this project.

1.1 Objectives

The basic objective of this project is to simply analyze a fashion image and predict how popular the trend will be. This idea seems clear and simple but as mentioned earlier: what is a popular fashion trend is abstract. To solve this question, we address the specific definition of what is good fashion. According to previous discussion, the trend of fashion is something which can help a company to know what kinds of fashion is popular or unpopular. Therefore, more specifically, we can say the trend of a fashion is approximately equivalent to the popularity of a fashion; the trend of fashion stands for how the popularity of the fashion. And if what is being analyzed is a trend, according to the statistic concept, this can be thought of as measuring the collective perception from a large amount of data. For example, if many people like a dress, then this fashion dress is deemed popular. Furthermore, if the whole population in the world likes a fashion dress, there is no doubt then it is good fashion. As a result of this reasoning, the aim of this project is to predict popular fashions based on finding fashion trends as defined by analyzing a large amount of data samples. To complete this project, the following objectives need to be met.

The objectives and their possible outcomes are illustrated as follows:

1.1.1 Collecting a large amount of fashion data

More data not only benefit finding more accurate trends but also provides a perfect condition to use machine learning algorithms. With the accumulation of data from social networks in recent years, data can be collected from fashion-related social networks. The data should generally include the following information: Firstly, the popularity of the each fashion as indicated by user voting. Secondly, the attributes or features of the fashion dress such as: the type, pattern, style, texture, color, brand, location where the fashion will be worn etc. Additionally, some supplementary information which may help to predict the popularity also needs to be collected. For this project the website *www.lookbook.nu*, which contains over two million fashion records, was chosen as the data source. For more data granularity, the original images of the fashion dresses and the users' information were also collected.

To achieve this, several web crawlers were designed. The sequences of data collection were carefully determined because of the time needed to collect data which may also contain much

useless information. A bad collection sequence slows the progress of the project and may cause other potential problems.

1.1.2 Data Management and Data Pre-processing

To utilize the collected data efficiently and effectively, it should be stored under a well-designed data architecture. Text and number data were stored in a database while the raw images were stored on a hard disk with a well-designed file path structure.

Besides this management, pre-processing on data was performed to ensure the data quality, format, scale etc. is suitable input for the machine learning model. More specifically, data quality contains three standards: no error data, no redundancy data, and no missing data. The data quantity was large, thus, these standards are hard to maintain but there must be some method applied to improve the quality of data. Moreover, the data was adjusted into a suitable format. Normalization and standardization were employed to speed up the convergence of the model. Other techniques were used depending on the real working conditions.

1.1.3 Information Extraction and Data annotation

The raw data usually not only contain noise and errors, but also conceal some important information. Raw images are typical data which contains abundant potential information, and almost all information of the fashion can be retrieved from the images directly. Therefore, we need to extract more useful information such as: color of the dress, style of the dress, the biological features (gender, race etc.) of the person in the image, or even the occasion of the image where it was taken. These kinds of extractions can be performed by humans, which will achieve the highest accuracy rate on the extracted information. Or it can be performed by computers dealing with large amount of data but with lower accuracy. For this project because of the large amount of data, we used computers to annotate the attributes in the images with human assistance for correction. This technique of computer annotation was discussed by Liu Ziwei et al. [6] [7]. As a result, building the data extraction model is critically important. This project's model is a CNNs [8] (convolutional neural networks) built and trained carefully using this paper as guidance. There may be some modifications depending on the real cases in this project. With this information in addition to the raw data collected from the social network, every data point may be annotated and combined in a suitable format to construct a dataset as a final training set for the prediction model.

1.1.4 Predicting the popularity of the fashion

With the prepared dataset, we used training to help the model predict whether a fashion in a picture is good or not. A multi-layer neural network was employed to finish this task. There are two reasons to use this technique. The first one is it has a good non-linear data fitting ability, and the second is that it may compute any function [9]. Too many dimensions (attributes extracted from fashion) used as inputs of the model produce a high level of complexity that many linear approaches cannot handle in addition to the large amount of data. The output of the prediction mark is a consecutive number, which may be handled as a regression process. There are many models that can perform this regression: linear regression, logistic regression, SVMs, and of course neural networks. Among these models, neural network is able to outperform other models when the data amount is large. For this project, the raw data contains more than two million pieces of information.

1.1.5 Visualization

It is important to visualize the model since this project intends to predict whether a fashion is good or not. Fashion is depicted as an image used as the input of the model, with the output of the model being a mark of the fashion to indicate popularity. This can be performed using a client-server type web application.

1.2 Risk Assessment

Table 1: Table of prioritized risks

Priority	Risk Identifier and Description
1	Poor performance of annotation CNNs
2	Poor convergence and over-fitting of the neural networks
3	Fashion website forbiddance to crawlers
4	Data loss
5	Fashion website shutdown
6	Slow training of neural networks
7	Missing data
Priority 1 is the highest priority	

IMPACT / PROBABILITY	LOW	MEDIUM	HIGH
LOW		5.Fashion website shutdown	4. Data loss
MEDIUM		3. Fashion website forbiddance to crawlers 6. Slow training of neural networks	1. Poor performance of annotation CNNs 2. Poor convergence and over-fitting of the neural networks
HIGH	7. Missing data		

Figure 1: Probability impact matrix before proposed solution

Problem Descriptions and Contingency plans

1. Poor performance of annotation CNNs

   The annotation network is used to extract information from the images, but it may produce many incorrect annotations. If these networks performed poorly, it will lead to low data quality and poor prediction accuracy. The theoretical performance of the annotation networks is high, so this problem can be solved by well designing and training the annotation under the guidance of the papers [6] [7]. After careful work, if the network still does not satisfy the requirements, the annotation process can be performed by humans, but it will take a longer time.

2. Poor convergence and overfitting of the neural networks

   Poor convergence of the neural network means poor performance after training. Over-fitting means the neural networks take over after the steps of training, but perform well only on the training dataset, not on other datasets. This situation is very common in neural network training, and it will directly affect the performance of the system. Nowadays, this problem can be solved by assuring good data quality, applying regularization or dropout [10] techniques, setting good empirical hyper parameters [11] etc.

3. Fashion website forbiddance to crawlers

   This problem is very common when using web crawlers to collect data, there are several ways to avoid this problem: reduce the frequency of data request, or using dynamic IP addresses etc.

4. Data loss

   Data loss means losing part or all the data which was collected from a website. Since this project uses a large amount of data, data loss is a problem which affects the progress of the project. Although this is a simple problem and seldom happens, but once it occurs, the project may need to restart from the beginning. It can only and easily be prevented by frequent data backup.

5. Fashion website shutdowns

   The website from where we collect the data may shutdown. This is unlikely, but once it happens, it cannot be fixed. In this case, we will need to change to another fashion website. We can accelerate the pace of data collection to reduce the impact of this problem.

6. Slow training of neural networks

   Because of the high demand of computing power during the network training, the training stage may take a long time. This is a major problem in previous neural network studies and applications development. However, with the development of the GPU (graphics processing unit), the training time can be reduced by using more powerful GPUs.

7. Missing data

   Since we are collecting millions of data, there may be some missing data during the collection. This will affect the whole system only slightly. To achieve better performance, improved design of the data crawler will reduce the occurrence of missing data.

Reassessment of risks after applying the contingency plan

After applying the contingency plan, one of the seven risks is completely eliminated while the remaining six risks are diminished.

Table 2: Table of prioritized risks after applying the contingency plan

Priority	Risk Identifier and Description
1	Poor performance of annotation CNNs
2	Poor convergence and over-fitting of the neural networks
3	Fashion website shutdown
4	Slow training of neural networks
5	Fashion website forbiddance to crawlers
6	Missing data
Priority 1 is the highest priority	

IMPACT / PROBABILITY	LOW	MEDIUM	HIGH
LOW		3. Fashion website shutdown 5. Fashion website forbiddance to crawlers	
MEDIUM	4. Slow training of neural networks 6. Missing data		1. Poor performance of annotation CNNs 2. Poor convergence and over-fitting of the neural networks
HIGH			

Figure 2: Impact and probability of risk after re-assessment

1.3 Summary

After a brief introduction of this project, Chapter 2 highlights the background information and its related works. Chapter 3 discusses several completed works, as well as important analyses and decisions in this project. In Chapter 4, some on-going experiments and future plans are discussed in detail. Finally, Chapter 5 offers a conclusion for the report.

2 Background and Related Work

In this chapter, detailed background information related to fashion are discussed. Moreover, some theories of machine learning and neural network are provided. The focus of the third section is related works of the computer science research on fashion analysis, as well as some related theories.

2.1 Fashion

Fashion is the popularity of style, especially in clothing which generally represent items such as dresses, pants or skirts, footwear, hats, makeup, glasses etc. It changes over time. Fashion has many attributes that are combined together to determine popularity. As clothing which plays a role of decoration in society, the obvious features of fashion can be easily detected by vision such as color, style, texture, pattern, print graph, size, category etc. It also contains some features which are not easily noticed such as material, culture metaphor, price, political and social class metaphor etc. This combined effect indicates what fashion is, but still there are many features which may affect the fashion: simply speaking "who wears what fashion on what occasion judged by what kinds of group of person under what time background." Fashion itself affects its value but also many other factors. Therefore, fashion is hard to judge as good or bad, but rather different persons may have different comments.

Fashion is highly complex. Few public researches can be found related to the fashion trend but it does exist. The first evidence is based on a common sense in our daily life. Today, it is easy to see in magazines or on the internet what is popular now or next season. Also, when you walk in the shop, the style of the clothes in the shop changes from season to season. There are always new popular types of fashion today and in the future. The second evidence is there are some famous trend forecasting companies, like WGSN (Word Global Style Network) [12], which provides fashion forecasting services. In fact many famous clothing companies, such as Nike [13], Adidas [14], Levi's [15], Coach [16], H&M [17] etc., use these kinds of services to guide their business development [18]. According to the WGSN website briefly provided [19], the trend analysis on fashion will be based on color, catwalk show, images, prints, graphics, CADs (Computer Aided Designs), retail intelligence, and an experts group. According to the two previous evidences indicated, there are trends in fashion, though it is hard to describe.

2.2 Machine Learning

Some brief background information of machine learning are introduced in the introduction In this section, more detailed information on machine learning technologies used in this project are provided.

2.2.1 Artificial Neural Networks

Artificial neural networks (ANNs), or neural networks, or multilayer perceptron (MLP) are similar concepts in machine learning theory. This model was first mentioned by Warren McCulloch and Walter Pitts in 1943 [20]. It was called preceptors at that time and was a mathematical-based model. ANNs is constructed by using a collection of units called neurons. These neurons lie layer over layer, and the neurons of the adjacent layer are connected to each other. The first layer of the network is used for data input while the last layer of the network generates data output. The ANNs is used for learning the regularity of the input data based on a mathematical model. It uses a loss function to measure the deviation of the output from ground truth and to learn is to minimize the lose function. The loss function is different from problem to problem and the most popular way to minimize the loss function is gradient descent [21] which uses mathematical approaches to minimize the loss function iteratively. Since ANNs work by reducing the loss function, the relationship between the input and output is not necessarily known and as previously mentioned, it has a very good ability to deal with very complex works. There are some drawbacks of ANNs. According to one paper [22], one of the major problems of ANNs is overfitting, which means the model only perform well on the training data. Another problem with ANNs is a time-consuming training period since it needs a large amount of computational resources.

2.2.2 Convolutional Neural Networks

With the development of deep learning in recent years, there are many deep neural networks models that are created to perform different jobs. Convolutional Neural Networks (CNNs) is one very famous model first proposed by Y. LeCun et al. in 1989 [23]. This model's major use is for image processing. Unlike the traditional ANNs, CNNs cancels the connection between the layers and replaces them with shared weights and to generate features maps layer by layer. The advantage is to reduce computation need and extract more representative features. Generally speaking, CNNs is performs well in extracting abstract and useful features

from images, and these higher level features can be used for classification or other jobs. In this project, VGG-16 [24] and Xception [25] are employed as the base to construct the model.

2.2.3 Tensorflow

With the development of deep learning in recent years, many deep learning frameworks have appeared making the study and implementation of deep learning easier than before. For this project, Tensorflow [26] which is produced by Google are the tools used to construct the neural network. Tensorflow is an open-source software library for machine learning intelligence, it provides a Python [27] API for implementation. It also allows a NVIDIA [28] GPU acceleration to speed up the training progress.

2.3 Related Work

Like other subjects of study, the study of fashion in computer science, more specifically relating to computer vision and machine learning, can be summarized in a progressive level by level structure. According to a brief abstract of study, there are three levels.

The first level of study focuses on clothing parsing which extracts the clothes in image. Basically, the focus in this level is a kind of image segmentation. Edgar Simo-Serra et al. [29] used CRF (condition random fields) [30] model to perform this work. And more recently Liu Ziwei et al. [6] [7] proposed the concept of fashion landmark and used CNNs [8] (convolutional neural networks) to improve the performance on the fashion detection.

The second level is to extract some higher level attributes from the clothes in image, like the type of the clothes, the style or pattern of the clothes, or even the texture of the clothes. In this level, the study focuses on the detail of the fashion. Liu Ziwei et al. [6] [7] used the CNNs to find attributes and utilize them to perform some batch mark like clothes retrieval, clothes classification etc. Edgar Simo-Serra et al. [31] also used the CNNs to learn the style of the fashion in the pictures.

Based on the first two levels, the third level of study has the closest relationship to this project. It does not only focus on the fashion itself, but uses fashion to produce more abstract outputs such as the occupation of the person who wears the clothes, or the trends of the fashion etc. Zheng Song et al. [32] proposed an occupation prediction framework using sparse coding. Ziad Al-Halah et al. [33] evaluated and forecasted the fashion style by

collecting the clothes purchasing data from Amazon (Amazon.com). Edgar Simo-Serra et al. [34] implemented a CRF model to predict the “fashionability” [34] of the clothes of people using over 140 thousand data points collected from a fashion website.

3 Design Approach

This project can be divided into three major parts: data collection, data processing and prediction implementation. In the first part, the crawler design, strategies for fetching and fetching content are discussed. In the second part, data management, analysis and normalization are indicated. Finally, the architecture of the prediction model, the finished training experiment and the training machine detail are discussed and illustrated in the third part.

3.1 Data Collection and Processing Design

The data from social networks always contain some redundant, irrelevant or harmful pieces. The useless data may not affect the learning process of the model negatively but will delay data collection. Since the duration for this project is one year, time is a very important element. As a result, data collection focuses on introducing some special strategies, designs and considerations.

3.1.1 Web Crawler Workflow

This project aims to collect and use data from fashion social networks to perform the prediction. First, a web crawler was used to collect data from the social network. The crawler uses the following three steps iteratively until it reaches a condition to stop. 1. Send a request to get a web page. 2. Parse the web page and extract data. 3. Store the data. What is different from a traditional crawler is that there were three types of crawlers (fetchPerson, fetchPost, fetchLook) used in this project to fetch the data. As depicted in Figure 3, the output of one type crawler is used as the input for the following crawler to fetch new content. The fetchPerson.js first generates the Person List of the website with the Person Data (Person means the users of the website). Based on the Person List produced by fetchPerson.js, the fetchPost.js fetches the Post List and Person Data from the web (Post means the fashion post on the fashion website). Person Data is used to update the previous output and the Post List as the input of fetchLook.js is used to fetch the Look Data with images (Look Data means the detail data of every post). The reason why the crawlers work in this sequence is the special design of *lookbook.nu,* in addition to using two fetching acceleration methods applied for this project which will be explained in the following chapter.

.

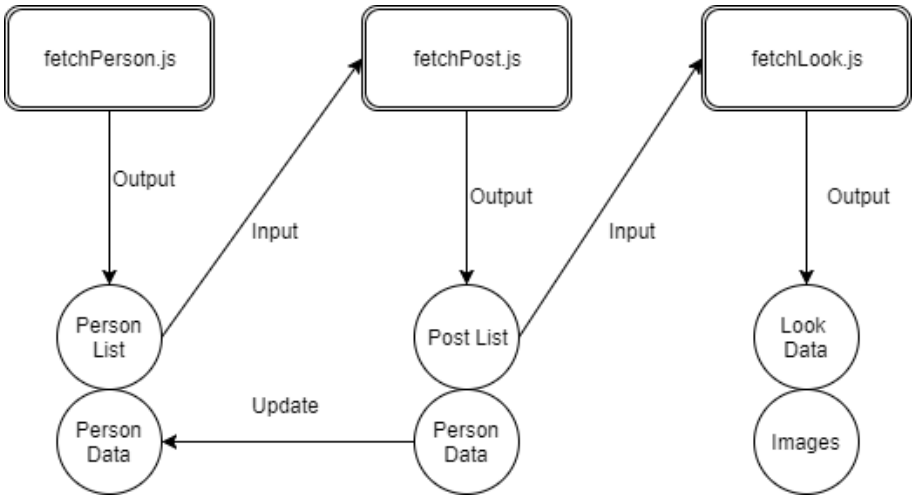

Figure 3: Crawler workflow

3.1.2 Basic Data Fetching Strategy

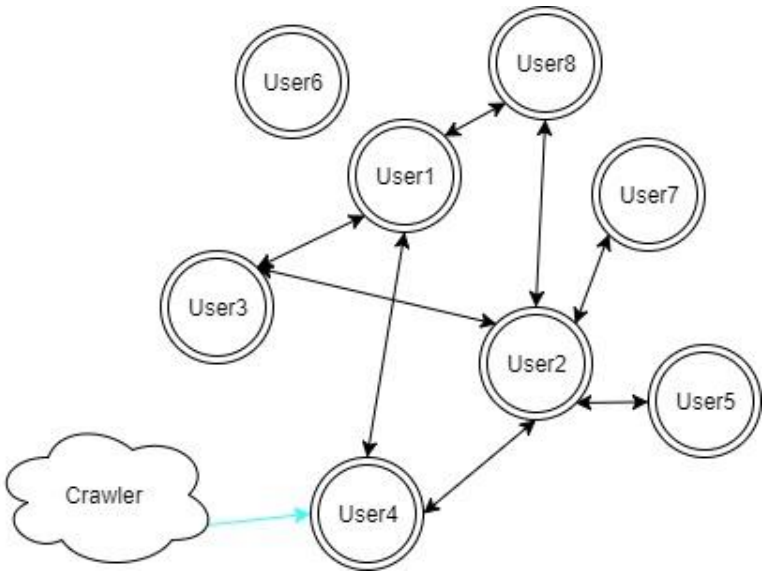

Figure 4: Common social network

As previously described, the three crawlers work in a sequential order. Normally, fetching data on a social network is time consuming because of the relationship of the social network. As shown in Figure 4, a crawler starts from one or many users to fetch the data. Crawlers usually get the information of the other person from the current fetching user, and after fetching the current user it will jump to one of the current person's related person to continue fetching. The only way for a common crawler in a social network site to know where to fetch data is to extract information from previous fetched users. This rule limits the efficiency of

the crawler in the social network because there may contain redundant user connections such as User2 in Figure 4, which will produce redundancy in the fetching. Moreover, there are some users, like User6, that do not have any relationship with other users. A crawler will not fetch these kinds of users. This may damage the integration of the data.

Fortunately, the crawlers working on *lookbook.nu* can avoid the drawback of the traditional crawler because this website gives every user and looks (the post on the website) a unique ID number, and these IDs are related to the URL of users or looks. Therefore, the crawler may traverse all possible ID numbers one by one. Also, the IDs of the users are not fixed long length numbers but start from small single digit numbers, while not consecutive but is still dense. These special features allow the crawler to avoid repeatedly fetching some users with abundant relationships while miss fetching some isolated users. Moreover, the system allows the collection to process at many terminals simultaneously as well as independently. The fetching strategy introduced is based on these special conditions of this website. Since the person ID exploration is from a small number to a large number, the determination of when to stop is important. And the stop moment in this project is simple, which is when the crawlers cannot find any user IDs in a large range, like 10,000 to 20,000. After introducing this special strategy, the workflow design of the crawler is shown in Figure 5.

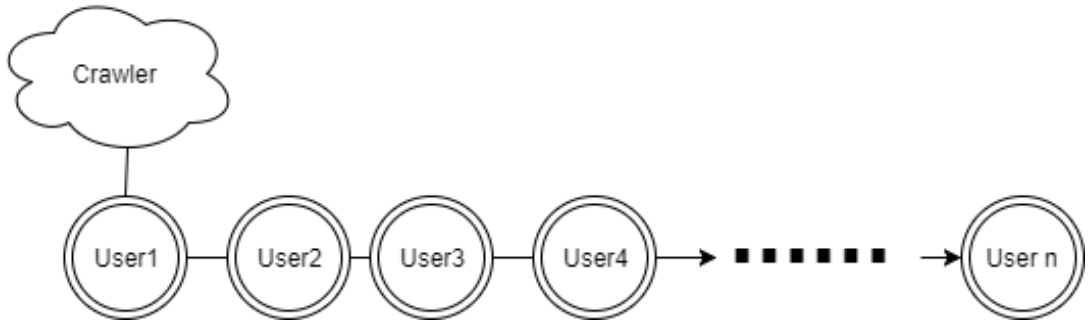

Figure 5: The workflow of crawler when fetching person data

This fetching strategy only speeds up the collection of users but not looks (images). The reason is when the crawler accesses the user with different ID numbers, it can only access the user's home page which may not contain whole links to the looks (images) posted by users. Luckily, the list of looks page may be accessed using the user ID which will be available after collection of users. If the crawler wants to collect all the links of looks, it may depending on the number of looks, need to send more than one requests to get the related web pages from the image links list. This is disappointing since the most time consuming job is the

downloading of looks. To overcome this weakness as well as speed up looks collection, the workflow in Figure 3 was designed. The collection of users will be launched initially, with the user ID and social network data of most users available after the collection. Based on this set of social networks data, the first acceleration method can be applied. The reason for the first acceleration method is to avoid the looks collection of some helpless users by analyzing the social network data, which will be discussed in Chapter 4.1.2. After the collection of users, the crawler (fetchPost.js) for collection of looks' link is launched. In this stage, the images will not be directly downloaded. Instead, they will be collected after some links of images has been stored. The reason is there is a limitation to the second acceleration method which utilizes multiple Raspberry Pi devices [35] to collect data simultaneously. Since each Raspberry Pi has limited storage, it is not suitable for collecting the images. Therefore, rather than sending another request to get the image, Raspberry Pi devices are used only in the collection of those image links, and which leaves the downloading of images to other suitable hardware. This second acceleration method will be discussed in Chapter 4.1.3. In conclusion, three crawlers (fetchPerson.js, fetchPost.js, fetchLook.js) will run on different machines and work together to speed up the collection.

3.1.3 Fetching Content

Since there is much information on the fashion website which may be used for the training of MLP with the data collection process beginning before the MLP design, fetching all information is the best way to ensure less duplication of jobs. As a result, the lists of the contents are under the fetching plan, each with a brief description, are shown in Tables 3, 4 and 5. Since the dynamic contents of the webpage, processed by using JavaScript [36] after loading the webpage, are hard to fetch in an efficient way plus these contents hold only little useful data, all the dynamic components of the webpage are not included in the fetching plan. Generally, there are three types of the data: User Data, Look Data, and Accessory Data. User data and Look data contain all information related to the users and looks. The Accessory Data holds ancillary information of images such as tags and items. Each look may own zero or more than one accessory. And because the Tag Data contains only the name of the tag, it is merged into the Look Data. The Accessory Data can then be named as Item Data.

Table 3: List of user data under fetching plan

Data Name	Description
`ID`	The unique ID of the user
`Name`	The name of the user
`Gender`	The user's gender post online (maybe unknown)
`City`	The user's city information post online (maybe unknown)
`Country`	The user's country information post online (maybe unknown)
`Fans`	The number of user's fans at the fetching moment
`Looks`	The number of user's posted looks at the fetching moment
`Heart`	The number of user's received hearts at the fetching moment
`Karma`	The karma value of user at the fetching moment
`Following`	The number of the people who were followed by the user at the fetching moment
`Topics`	The number of the topics owned by the user at the fetching moment
`Comment`	The number of the comments at the fetching moment
`CKarma`	The comment karma value of user at the fetching moment
`Since`	The timestamp when the user registered the website
`Position`	The position of user (Only OG and Member)
`Views`	The view time of user's profile at the fetching moment

Table 4: List of look data under fetching plan

Data Name	Description
LookID	The unique ID of the look
Title	The title of the look
Hype	The hype value, which means how many people like this look, of the look at the fetching moment
Imagelink	The URL link of the look's image
PostTime	The timestamp when the look was posted
UserVisit	The number of users who visited the look
Description	The description of the look
PersonID	The ID of the user who owns the look
Items	The items in the look. There may be zero or more than one item in the look, and each item may hold brand, name, category information as well as the position information in the image. See detail in the Table 5.
Tags	The tags name in the look. There may be zero or more than one tag.

Table 5: List of item data under fetching plan

Data Name	Description
Left_dist	The distance from the left side of the image
Top_dist	The distance from the top side of the image
ItemName	The name of the item
Brand	The brand name of the item
Category	The category name of the item
LookID	The look ID which holds this item
CategoryID	The categroy ID of the item
RealID	The real ID of the item
DisPlayID	The display ID of the item

3.2 Data Management and Analysis

3.2.1 Database Structure

Although the project's objectives is not to provide a high performance service for fashion data and image management or retrieval, building a robust database to store the data is still necessary because it is a good way to efficiently manage and analyze data. The database structure is illustrated in Figure 6.

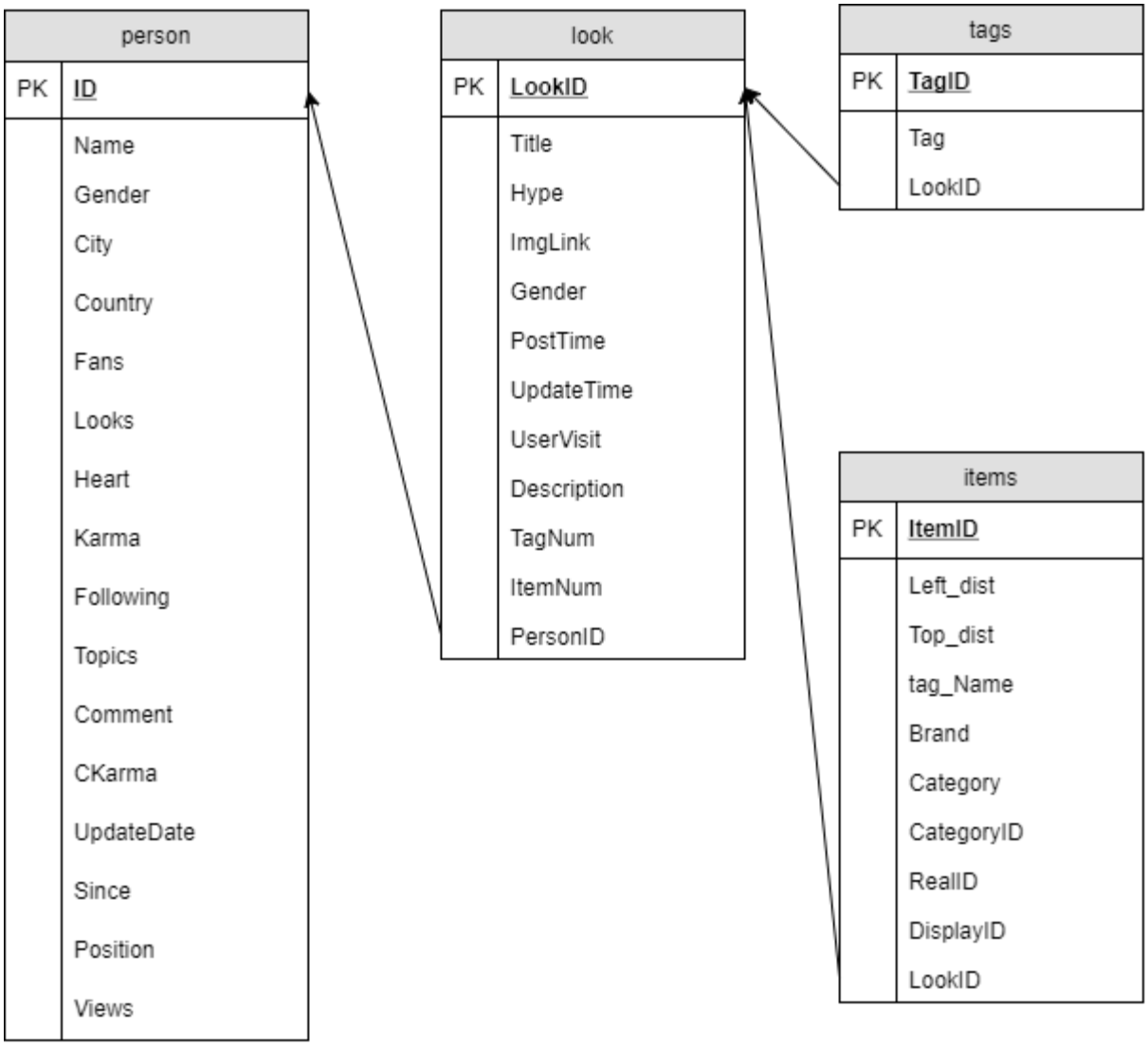

Figure 6: Database structure

3.2.2 Data Pre-processing and Normalization

Images are also used as data for training. For this project, to save the limited computation resources, the input image size was set as 224 x 224. Therefore, every image was resized to 224 x 224. During the reshaping, the aspect ratios of original images were unchanged, with black dots used as fill in to satisfy the required shape of pictures if necessary.

The time data is in timestamp format, which is a very large number. The seconds are very detailed in time dimension, but there is no need to use a detailed description of time. Therefore, the time data was transferred into the number of months from the time when the fashion website was firstly launched, which was March 29, 2008. During the processing, all months contain 30 days. The longest time descriptor after transfer was around 114.

The data ranges are different from attribute to attribute, with some that are quite large and others that are small. This situation may damage the convergence speed and quality of MLP. Thus, a data normalization process was used to rearrange the data to a unified range, like 0 to 1. The aim of normalization was to make the training job easier by making small or large input data within the same level. Therefore, to make the data more on the same level, it was remapped into the range from 0 to 1 with an exception of the values of the time data, of which the largest data was over a hundred. Since data is easier to manipulate when the levels are comparable, the time data was divided by 100 to keep its number level consistent with other data. The hype, which is the final fashion mark, was also rescaled into the range of 0 to 1, however it can also be used as the range of 0 to 100 by multiplying 100.

There are several normalization methods to reshape data. Linear function, Log function, Arctan function are common functions used to for normalization. Linear function follows the following formula:

$$f(x) = \frac{x - Min}{Max - Min}$$

Where *Min* and *Max* are the maximal and minimal value in the dataset.

Log function follows the following formula:

$$f(x) = \frac{log_e^{(x)}}{log_e^{(Max)}}$$

Where *Max* is the maximal value in the dataset.

Arctan function follows the following formula:

$$f(x) = \frac{2Arctan(x)}{\pi}$$

Where *Arctan(x)* is arctangent function.

Sigmoid function follows the following formula:

$$f(x) = \frac{1}{1 + d^{(m-x)}}$$

Where *d* is a constant which should be adjusted according to the dataset and the *m* is the median value of the dataset. In the hype data normalization, *d* is 1.1 and *m* is 70.

3.2.3 Idea of Removing Top Part

Data normalization is a good method to reduce or remove outliers from the dataset. However, some normalization methods, such as Log or Linear which utilizes the max and min value, has less ability to reduce the effect of the outliers. Moreover, the "good" of the fashion is quite subjective when the popularity of those fashions are all quite high. In this situation, it is hard to distinguish which fashion is better from those "good" fashions. In the view of data statistics, this can be seen as the outlier data of the dataset. Therefore, there was no need to distinguish them, which means that they are all good and no further action was needed to evaluate which are better among them. Therefore, the top 5% or top 1% data can be determined as the max value with no need for normalization. The remaining 95% or 99% may be used for normalization.

3.3 Prediction Model and Model Training

3.3.1 Overview of Prediction Model Structure

The abstract structure of the model is shown below in Figure 7. A good model is important for the performance of the system. The model described contains only two subcomponents: the Feature Extraction Model and the Regression Model. The workflow of this model begins with the Feature Extraction Model which takes a fashion image as input, and produces some representative features of the fashion for input into the Regression Model. Combined with other supplementary data from fashion website, the fashion features are processed as the training input for the Regression Model which predicts the mark of the fashion after several training iterations. Since this is an abstract model, there are several implementation methods. One method is to build the Feature Extraction Model and Regression Model separately. Another plan is to mix and train the two models simultaneously. It is hard to determine which concept will perform better given that fashion is so abstract. Therefore, comparing several models may be the best option and then choosing best performing model.

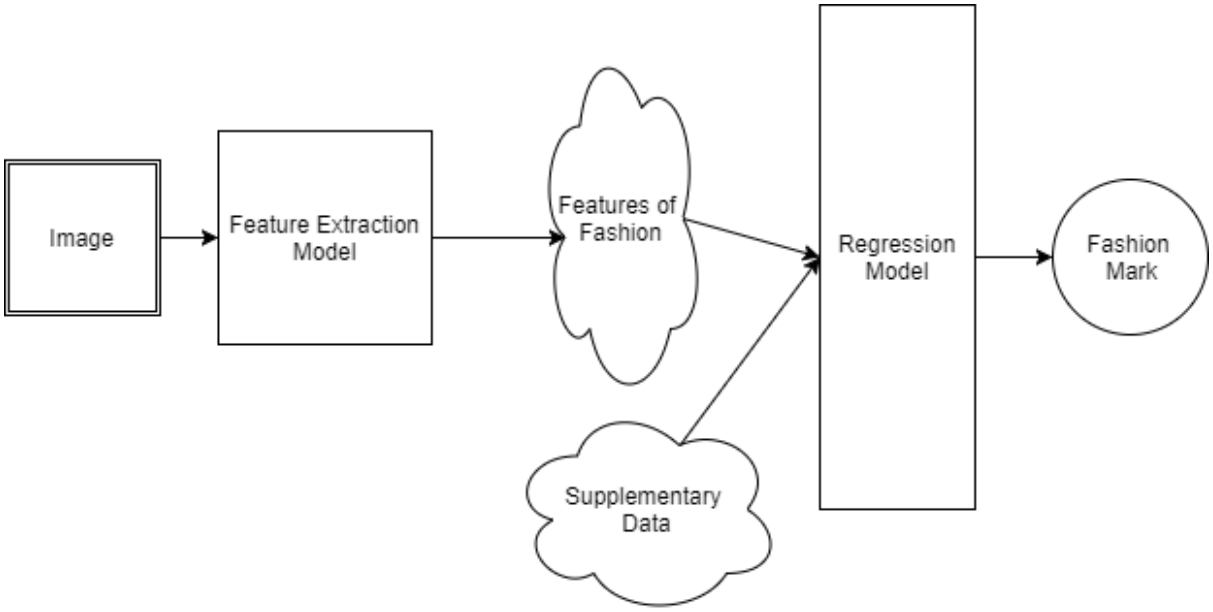

Figure 7: Abstract structure of model

3.3.2 Four Possible Plans of Prediction Model

In this project, several model combination plans are proposed. The details of the four plans are listed in Table 6. All four plans used a Regression Model based on MLP which are different only in the number of hidden layers and the number of neuron units in each layer. The Feature Extraction Model used was either the VGG-16 or Xception CNNs. The first three models were separately built models, of which the Feature Extraction Model was the VGG-16 CNNs model pre-trained on the DeepFashion dataset or a portion of the DeepFashion dataset. The reason for using the DeepFashion dataset is that it is a well-collected fashion related dataset. Therefore, the CNNs pre-trained on DeepFashion dataset should have the ability to predict fashion related features. To simplify the implementation process, the model structure (VGG-16 plus a special landmark pooling layer) proposed in the original paper of DeepFashion [6] was not used. Instead, the VGG-16 CNNs was trained directly with the dataset. This may cost some performance issues since the structure is simplified. To compensate the cost of performance, Batch Normalization [37] was used to improve the model's capability.

Table 6: Proposed plans

No.	Feature Extraction Model		Regression Model
1	VGG-16 trained on DeepFashion (Only fashion landmark)		Multi-layer Perceptron
2	VGG-16 trained on DeepFashion (Full Model)	Unreadable Features	Multi-layer Perceptron
3		Readable Features	Multi-layer Perceptron
4	Xception CNNs + Multi-layer Perceptron		

Among these four plans, the second and third plan used the same VGG-16 CNNs for Feature Extraction, but the two plans utilized output from different layers of CNNs. The second plan used the output of final pooling layer of CNNs which is not readable to humans, while the third plan used the final output from the whole neural networks which is also considered as the output from the final fully connection layer, that is readable to humans.

3.3.3 Architecture of Neural Networks

As indicated in the previous section 3.3.2, the Feature Extraction Model for all four plans was built based on CNNs. Here two types of CNNs, VGG-16 and Xception, were employed for building the Feature Extraction Model. Normally, the CNNs, including VGG-16 and Xception, is considered as the combination of convolutional with pooling layers and fully connection layers (Multi-layer Perceptron, MLP). Therefore, to simplify the wording and reduce confusion in this report, the convolutional with pooling layer was abbreviated into the CPLs (convolutional with pooling layers) with the fully connection layer simplified as the MLP (Multi-layer Perceptron). However, neither CPLs nor MLP of the CNNs can be trained individually. Thus, the DeepFashion CNNs will be trained using a whole (including CPLs and MLP) model. After training, the CPLs part of DeepFashion CNNs will be used

individually for the training of the fashion mark model. The detailed process are discussed in the next topic, with the detailed structure of the CPLs and MLP illustrated below.

The structure of CPLs of VGG-16 CNNs indicated in Figure 8 is used to build the Feature Extraction model for the first three plans. In the figure, each box except “image” and “features” contained five rows, representing the “name and number of layer”, “receptive field of filter” or “stride”, “number of output feature maps”, “activation function” or “pooling method”, and “size of output feature maps” respectively. In addition, the “conv” and “pool” represent the convolution and pooling layer. Moreover, the “elu” and “max” stand for the ELU (Exponential Linear Unit) [38] function and max pooling operation. As what indicated in the figure, the CPLs was able to transfer a 224×224 matrix (image) into a 7×7×512 matrix (features), which is called feature extraction. Though the matrix size was reduced by one-half after feature extraction, the image becomes much more representative information. Therefore, the regression model (MLP) can perform better by using the features rather than using the raw image directly. The CPLs holds the ability for feature extraction after training, and the CPLs will achieve different levels of feature extraction ability when trained with different datasets or performing different jobs.

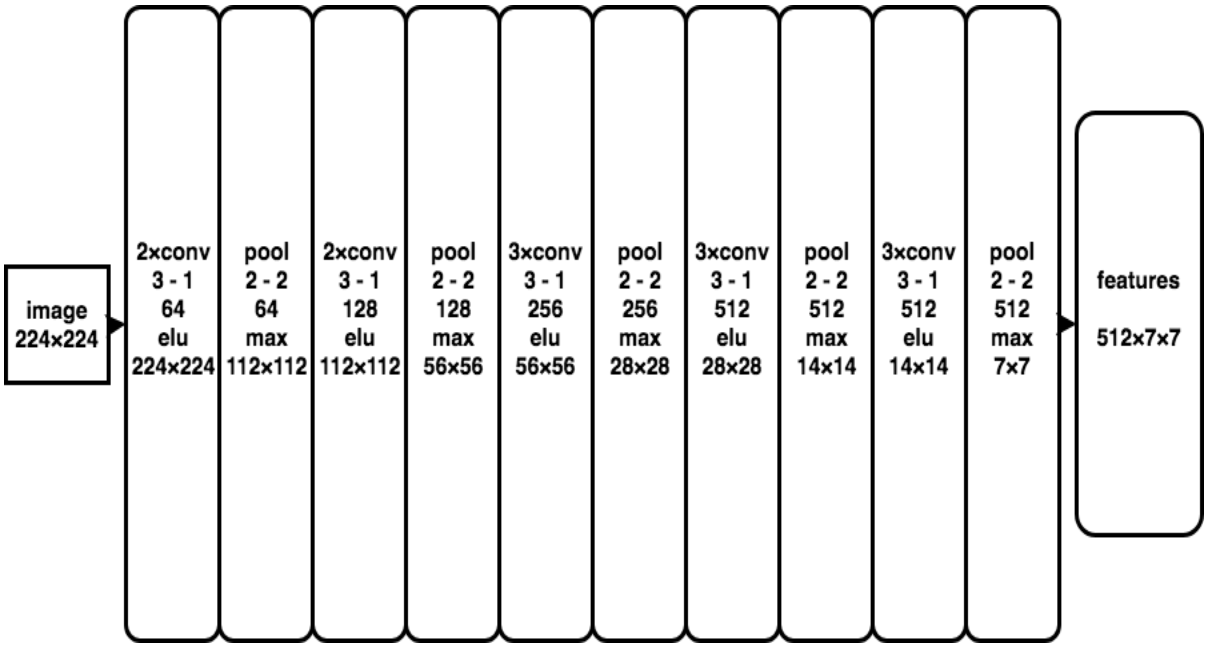

Figure 8: Structure of VGG-16 CPLs

The Xception structure shown in Figure 9 was used for plan four. Following the idea in the original paper of Xception, this model was constructed using three flows: Entry Flow, Middle Flow and Exit Flow. "Conv", "ELU", "MaxPooling", "SeparableConv", and

"AveragePooling" stand for convolution operation, ELU activation, max pooling operation, separable convolution operation, and average pooling operation respectively. The 224×224 matrix (image) was processed through the Entry Flow first, and then enter the Middle Flow. The Middle Flow process showed in the figure was repeated eight times. Finally, the data goes through to the Exit Flow and transferred into a 2048 features vector. The MLP is also shown in Figure 9, which contained two layers with 1024 units in each layer. There was only one final output which is the mark of fashion, and it goes through the Sigmoid activation. The reason for using the Sigmoid function was to rescale the output that is the hype to a range from 0 to 1. To match the output with the label, the output of model is also scaled. This action is also applied to Plans 1, 2 and 3 as indicated below.

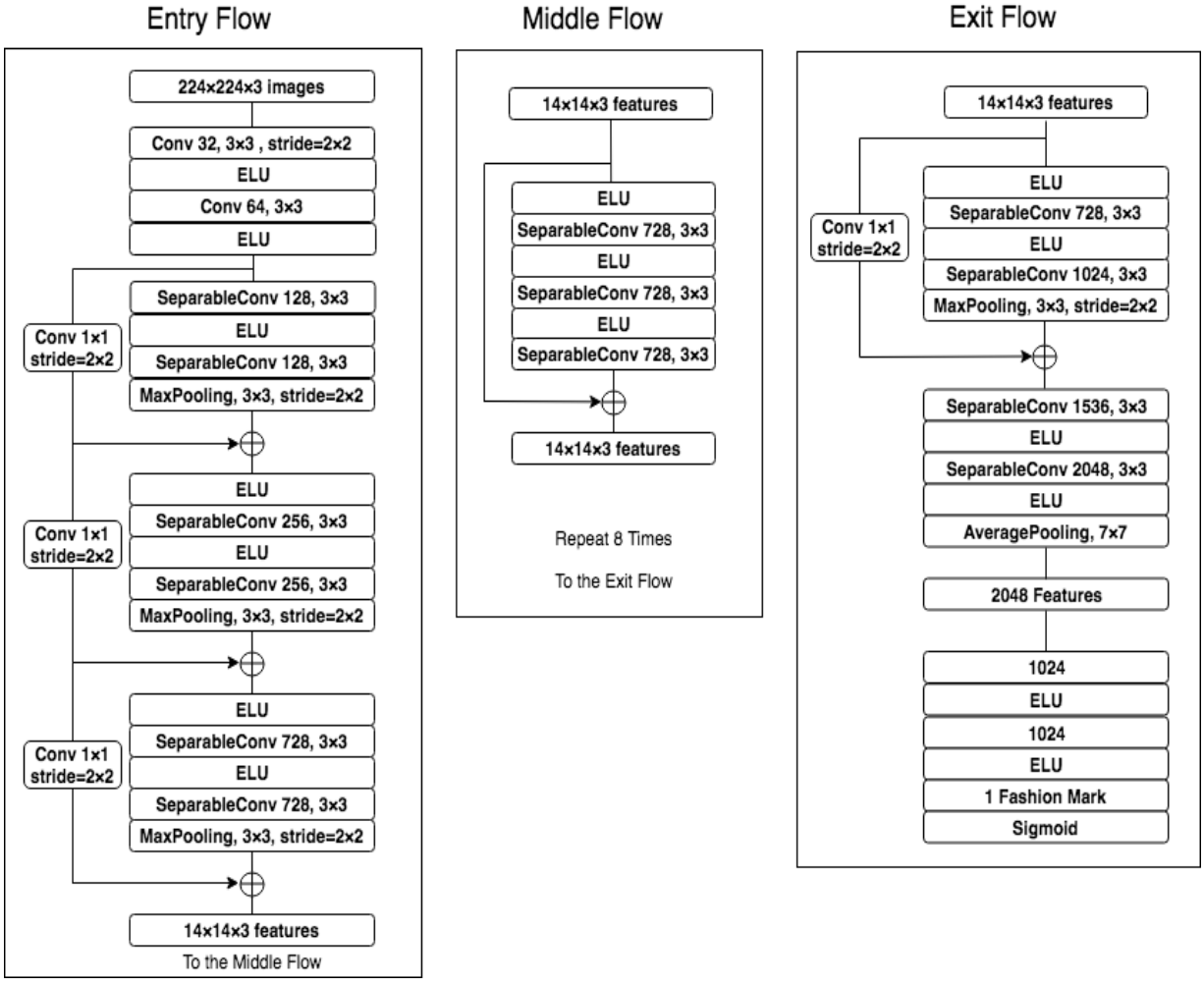

Figure 9: Structure of Plan 4 (Xception with MLP)

Different plans hold different MLP structures. The structure of MLP for plans 1 and 2 are shown in Figure 10. In the figure, the input to the MLP was the combination of features from CPLs of VGG-16 and the supplementary data of 18 social network attributes. The box is the

MLP and there were three rows, which indicates the “number of fully connection (fc) layers”, “activation function” and “number of neural units”. The final mark was the output of the MLP rescaled using the Sigmoid activation into the range from 0 to 1.

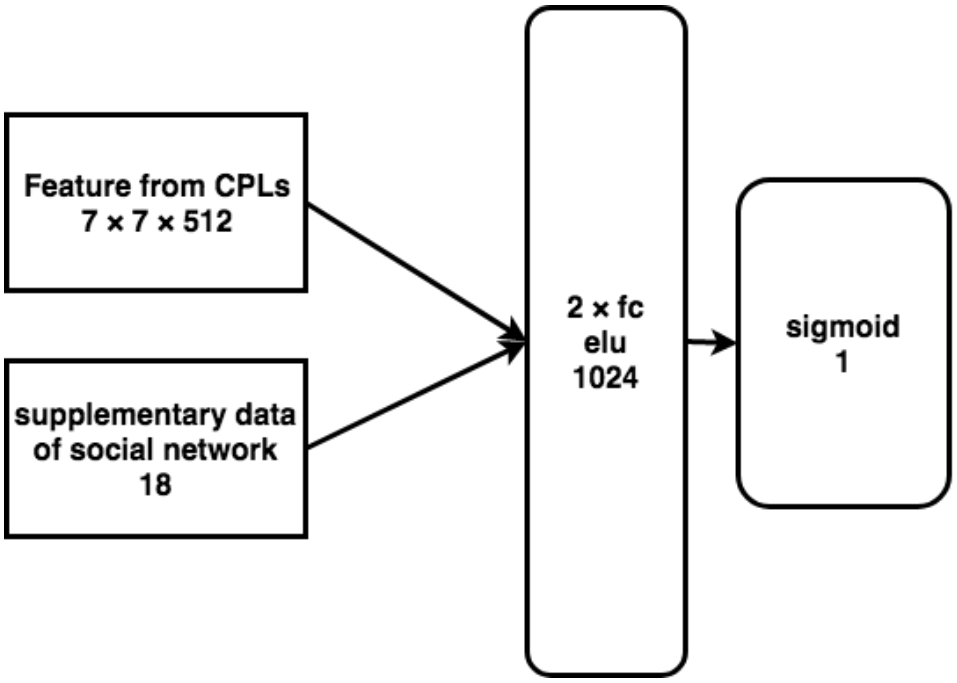

Figure 10: Structure of MLP of Plans 1 and 2

Figure 11 shows the structure of the MLP in plan 3. There is little difference except the input format. The input for plan 3 was the final output of the DeepFashion (full) model which includes: 1000 attributes, 50 categories, 32 values for the coordinates, and 8 visibilities of fashion landmarks. Again, the final output was remapped using the Sigmoid activation from 0 to 1.

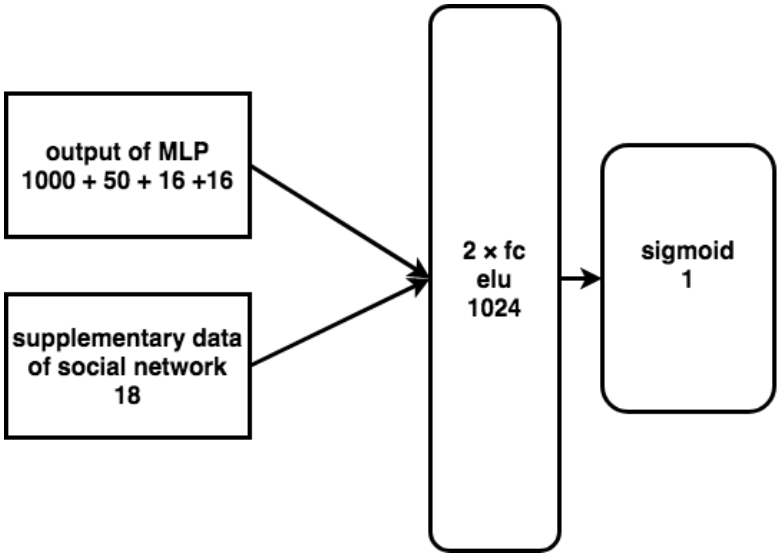

Figure 11: Structure of MLP of Plan 3

VGG-16 trained on DeepFashion dataset is shown in Figure 12, with the only difference in the input. The VGG-16 trained on DeepFashion dataset does not require supplementary data from the social network. The input was the features from CPLs only and the outputs are fashion landmarks, visibility, categories and attributes which is labelled in the DeepFashion dataset.

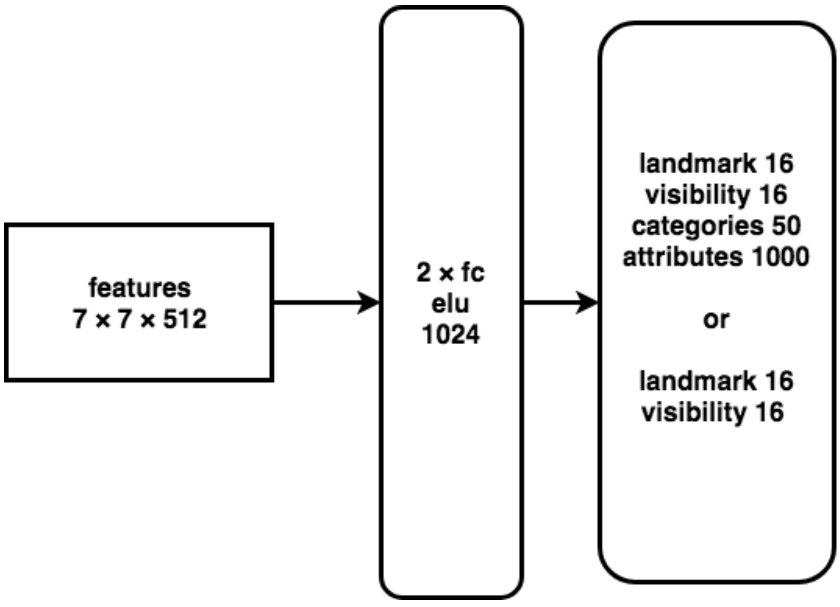

Figure 12: Structure of MLP of VGG-16 CNNs trained on DeepFashion dataset

3.3.4 Overview of Training Procedures

In total, there were six neural networks that was trained to build the four proposed plans.

Table 7: Six neural networks needed to be trained

No.	Name of Neural Networks
1	CPLs (&MLP) of DeepFashion (VGG-16 trained on fashion landmark only)
2	CPLs &MLP of DeepFashion (VGG-16 trained on Full Model)
3	MLP of Plan1
4	MLP of Plan 2
5	MLP of Plan 3
6	CPLs & MLP of Plan 4 (Xception with MLP part)

The six neural networks are listed in Table 7. Among the six networks, Network 1 is the pre-requisite for Network 3; Network 2 is the pre-requisite for Networks 4 and 5. As discussed previously, the CPLs and MLP of Xception CNNs cannot be trained separately. Since this project does not require the MLP part of the Network 1, the MLP for Network 1 is indicated by the brackets. However, since the CPLs cannot be trained without the MLP, it was trained in order to complete the CPLs sections of the networks. The detailed training procedures are presented in Figures 13, 14, & 15.

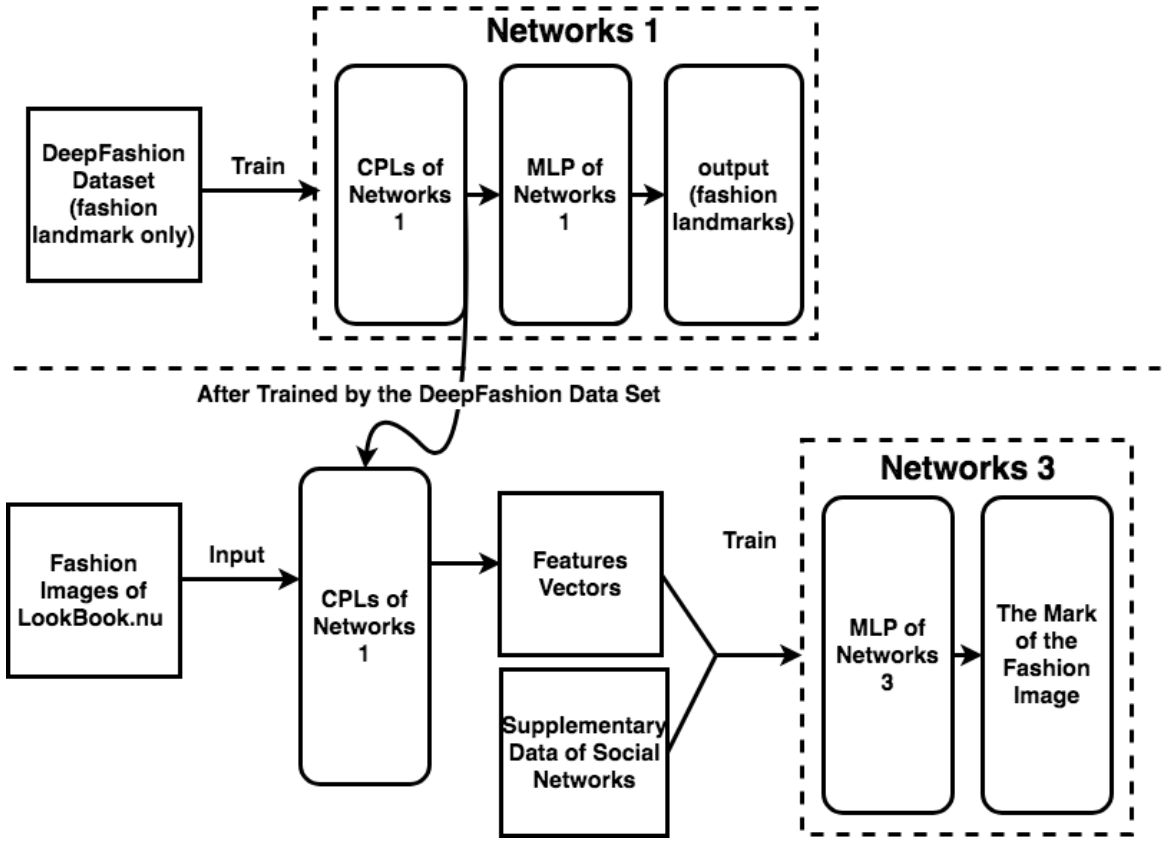

Figure 13: Training procedure of Networks 1 and 3

Figure 13 displays in detail the training of Networks 1 and 3 used in plan 1. As described previously, Network 1 was trained using the dataset from DeepFashion. This DeepFashion dataset contained only the fashion landmarks label, and Network 1 has the ability to predict the landmark of the full body fashion depicted via images. After training in Network 1, the CPLs parts of Network 1 performed the Feature Extraction model of plan 1. The fashion images from *lookbook.nu* are processed through the CPLs of the network to become a 512×7×7 features vector. This feature vector was combined with some supplementary data from the social network (lookbook.nu) and used as training data for Network 3. As a result, the actions of Network 1 must be completed before use by Network 3.

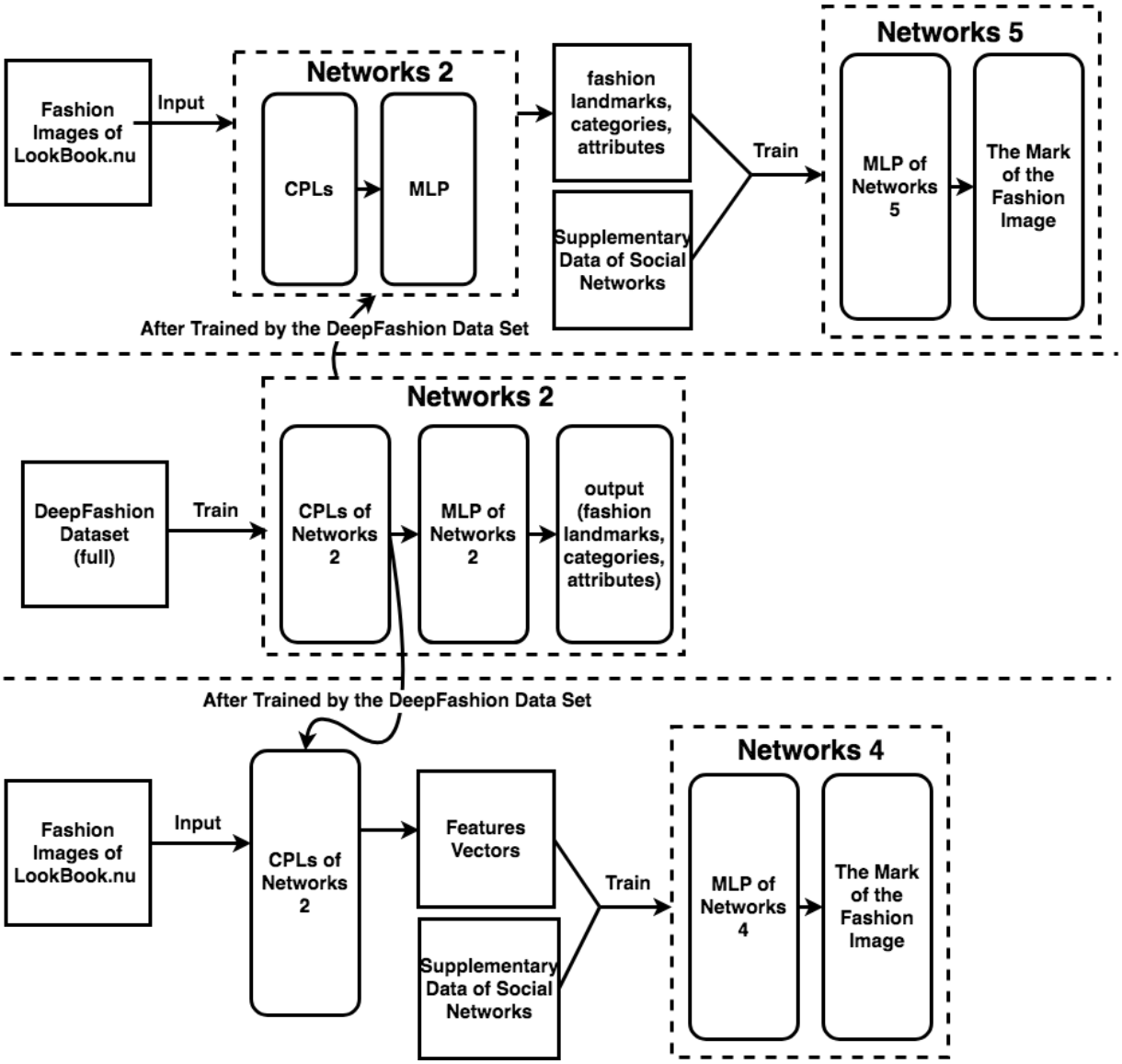

Figure 14: Training procedures of Networks 2, 4 and 5

Figure 14 indicates the training procedures for Networks 2, 4 and 5. These networks are the major components of plans 2 and 3. Network 2 was trained by using the full DeepFashion dataset which contains the labels of fashion landmark, categories and attributes. After training, the network was able to produce the fashion landmark, categories and attributes of a fashion image. Both plans 2 and 3 utilized Network 2 for the Feature Extraction models, but the two plans uses different parts of Network 2. For plan 2 which combines Networks 2 and 4, the CPLs of the Network 2 was used to transfer the fashion images to the features vectors as part of the training data for Network 4. For plan 3, it utilized Network 2 as the Feature Extraction model. Every fashion image from *lookbook.nu* was used as input into Network 2 to produce the final output of the Network 2, containing the attributes, categories, and fashion landmark

(coordinates and visibility) of the fashion image. This final output of the Network 2 is readable to humans, and was combined with supplementary data from the social network and used as training data for Network 5. Therefore, the Network 2 process must be completed before the training of Networks 4 and 5.

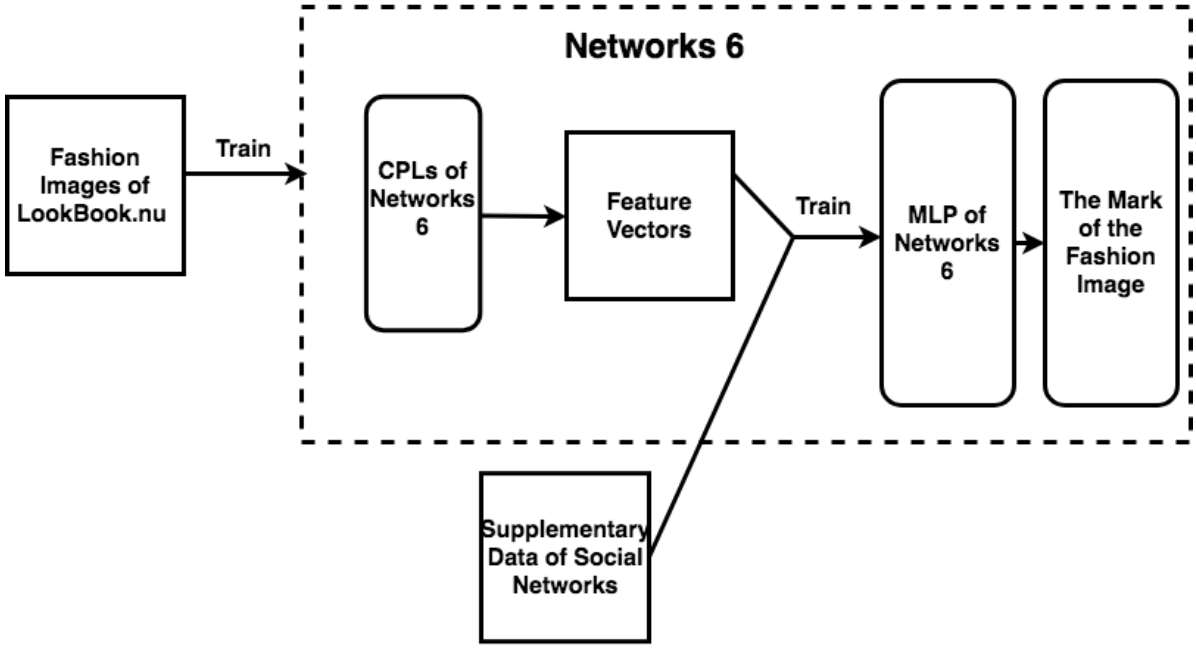

Figure 15: Training procedure of Networks 6

Figure 15 depicts the training procedure of Network 6, a major component of plan 4. Compared with the first three plans, plan 4 does not pre-train any Feature Extraction model. Instead, the fashion images from *lookbook.nu* are directly entered into Network 6 as training data. During the training step, the CPLs processed the image to produce the feature vector. The feature vector are combined with supplementary data, then entered in the MLP to produce the final mark. The final mark was compared to the real ground truth label of the fashion image, while performing a gradient descent optimization to modify the CPLs and MLP of Network 6 simultaneously. Plan 4 is the simplest design concept - it does not require a pre-trained model. It can be used as long as the data is available.

3.3.5 Measurement Methods for Fashion Mark Prediction

Two methods were designed to measure the performance of fashion mark prediction: Measurement by Output Mark (MOM) and Measurement by Top Percentage Rank (MTPR). The explanation of two methods is indicated below.

Measurement by Output Mark (MOM)

This method measures the absolute difference between the Ground Truth Mark (0-100) and Network Prediction Mark (0-100). If the absolute difference is smaller than a specific value (Bin Size/2), the prediction is deemed as correct. For example, when Bin Size is 20 and Ground Truth is 15, the Network Prediction is correct if it is between 5 and 15. The accuracy rate on the test set is the result of the MOM.

Measurement by Top Percentage Rank (MTPR)

This method measures the absolute difference between the Top Percentage Rank of the Ground Truth in the Ground Truth List, and Network Prediction in the Network Prediction List. If the absolute difference of the Top Percentage Rank is smaller than a specific percentage value (Bin Size/2), the prediction is deemed as correct. For example, when Bin Size is 20% and Ground Truth is ranked at 15% from top, the Network Prediction is correct if it is ranked between 5% and 15%. The final accuracy rate on test set is the result of the MTPR.

3.4 Visualization

As planned in the beginning, a client-server web application was designed to demonstrate the output of the project. Since this was not a major component of this project, the web page is only implemented on the localhost. The web page was designed by using HTML [39], CSS [40] and JavaScript. In the back-end, the server was implemented by using an easy-to-use Python framework Flask [41]. The reason why Flask was chosen was the neural network in Tensorflow was also trained using Python, which means it was much easier to invoke a neural network model in Python compared to other languages. Moreover, Flask can build a web server quickly and easily to save time in this project.

4 Implementation

4.1 Data Collection

4.1.1 First Type Crawlers Implementation

As discussed in the Chapter 3.1.1, there are three major crawlers (fetchPerson.js, fetchPost.js, fetchLook.js) which performed different jobs for the fetching strategy designed in Chapter 3.1.2. All three types of crawlers were implemented using the language of JavaScript (JS) and the Node.js [42] which is a JavaScript runtime built on Chrome's V8 [43] JavaScript engine. There are two reasons why Node.js and JS were chosen. The first reason was the simple and efficient features on both development and performance. In implementation, only one website was used to gather data, and a large framework may not be suitable because of complex development. Compared with Python which is another popular web crawler language, both provided an easy-to-use HTML parsing library. But Node.js is much faster when dealing with large I/O operations due to its asynchronous feature. The second advantage is that Node.js can be easily deploed on different platforms by only simply copying the small source engine code and some module libraries. These features were suitable in this project because several Raspberry Pi devices were used to accelerate the collection. In the implementation, the modules used to send HTTP request, perform local I/O, request image file and parse HTML document were http [44], fs [45], request [46] and cheerio [47] respectively.

Following the workflow indicated in Figure 3, fetchPerson.js is discussed. This crawler was used for fetching the data of persons (users) in the social networks (*lookbook.nu*). The main function of the code is shown in the Figure 16. The basic code logic was the crawler prepared some initial variables in the log record for later use. It looks for the last check point of the collection, which means the crawler can be interrupted and recovered to continue its job. It determines the number of iterations given the number of user IDs. In the loop, there were other supplementary operations which supported the process in log record, checked points saved and provided anti-blocking function. In the side loop, the crawler saved a check point every 10 iterations, stored data to a new file every 10,000 iterations, generated a new error data log file every 50,000 iterations, slept 1.7 seconds for every iteration and 20 seconds for every 1000 iterations to avoid being blocked by the website.

```
async function startFetchPerson(){
    var personHeaderUrl='http://lookbook.nu/user/';
    var JobDate=new Date().toLocaleString();
    var requestTimes=0;
    var whichPersonID=popLastPersonID();
    if(whichPersonID==-1){console.log('Last Person File crash!');return;}
    console.log(JobDate,":Job Starts",'Start Person ID:',whichPersonID);
     while(whichPersonID<7500000){
        var isPersonRes=await fetchPerson(personHeaderUrl+whichPersonID,whichPersonID);

       if(isPersonRes)JobsDone++;
        whichPersonID++;
        requestTimes++;
        if(requestTimes%10==0){
            updateLastPerson(whichPersonID-1);
            if(requestTimes%1000==0){
                if(requestTimes%10000==0)fileName=Math.ceil(Date.now()/1000);
                if(requestTimes%50000==0)errorFileName=Math.ceil(Date.now()/1000);
                await sleep(20000);
            }else {
                await sleep(1700);//1.7s
            }
        }
     }
     console.log('Jobs End---- Start:',JobDate,' End: ',new Date().toLocaleString(),'.');
     console.log(JobsDone,' Persons collected in ',requestTimes,' requests.');
}
```

Figure 16: Main function of first type crawler

4.1.2 Second Type Crawlers Implementation

After the introduction to the first crawler (fetchPost.js), the second crawler is discussed, and its main function is showed in Figure 17.

```
function start(){
fs.readdir(personListPath, async function(err,files){
        if(err){
            console.log(new Date().toLocaleString()+':Start()');
            return;
        }
        console.log(new Date().toLocaleString()+' Jobs Start!');
        for(var i=0;i<files.length;i++){
            var personListData=await preparePersonList(files[i]);
            for(var PID of personListData){
                var startStamp=Math.ceil(Date.now()/1000);
                //console.log('PersonID:',PID);
                var looksNum=await fetch(PID);
                totalPostNumDone=totalPostNumDone+looksNum;
                if(totalPostNumDone>3000){
                    totalPostNumDone=0;
                    postListFileName=Math.ceil(Date.now()/1000);
                }
                await sleep(1000);//Each Person
                //console.log('Cost:',Math.ceil(Date.now()/1000)-startStamp);
            }
            personFileName=Math.ceil(Date.now()/1000);
            if(i%10==0&&i!=0){
                personErrorFileName=Math.ceil(Date.now()/1000);
                postListErrorFileName=Math.ceil(Date.now()/1000);
            }
            fs.unlink(personListPath+'/'+files[i],
             function(err) {if(err)console.log(new Date().toLocaleString());});
            console.log(i+'/'+files.length+' '+new Date().toLocaleString());
            await sleep(5000);//Each PersonList File
        }
        console.log('Done:');
    });
}
```

Figure 17: Main function of second type crawler

This crawler prepared the timestamps for log record, and fetched the person list file which was collected by the first type crawler. It loaded only one person list file per each iteration of the first loop and deleted it after finishing the collection of this file. The crawler stopped when all files are deleted. In the second loop, it fetched every user updated information in the file, as well as all look ID of the user. After fetching a user or finishing a whole file, the crawler sleeps to avoid being blocked. The Crawler also counted the number of looks, which have been fetched by it, in order to start a new the store file when old one is too big. This action was useful in preventing the size of a single file from becoming too big.

4.1.3 Third Type Crawlers Implementation

The final type of crawler (fetchLook.js) was used to fetch the image of look and related information. The workflow is similar to the previous two types of crawlers and its main function is displayed in Figure 18.

```
function start(){
    console.log('Start:'+new Date().toLocaleString());
    fs.readdir(postListPath, async function(err,files){
        if(err){
            console.log(new Date().toLocaleString()+':Start()');
            return;
        }
        //console.log(new Date().toLocaleString());
        for(var i=0;i<files.length;i++){
            //console.log(files[i]);
            var postListData=await preparePostList(files[i]);
            //console.log(postListData[postListData.length-1]);
            for(var postID of postListData){
                //var startTime=Math.ceil(Date.now()/1000);
                var lookData=await fetch(postID);
                await sleep(1000);
                //console.log('Using time:'+(Math.ceil(Date.now()/1000)-startTime));
            }
            if((i+1)%10==0)errorLooksFileName=Math.ceil(Date.now()/1000);
            looksDataFileName=Math.ceil(Date.now()/1000);
            fs.unlink(postListPath+'/'+files[i]
            ,function(err) {if(err)console.log('Delete File Error:',new Date().toLocaleString());});
            await sleep(2000);//Each Post List
            console.log('Progress:'+(i+1)+'/'+files.length+' '+(new Date().toLocaleString()));
        }
        console.log('Done:'+new Date().toLocaleString());
    });
}
```

Figure 18: Main function of third type crawler

After preparing data for log record, the crawler returns to the first loop to load the look list collected by the second type crawler. For each iteration of first loop, the crawler loaded only one look list file, and entered the second loop to collect all every looks' data in the file. After finishing the collection of a look list file, the crawler deleted each file, and will stop when all look list files are deleted. Sleep is executed after fetching each look or finishing a file to prevent blocking from website. As in the second type, the third type of crawler also counted the number of looks collected and switches the store file to prevent large size issues.

4.1.4 Store File and Data Format of Crawler

Three type crawlers need to store huge amount of data from internet. Therefore, the method of storing data after fetching is important. There are three main considerations: 1. Data should be stored in a simple way (easy to write and store), since these file are temporary and will be restored to database after collection. 2. Because there may some data are not considered at the early stages of design, the data format should be extendable. 3. The format of data and file should not consume large storage space since some stages of collection work on the Raspberry Pi which has limited storage. As a result, the JSON [48] data format and Text (.txt) file format were chosen as the store standard. JSON is an extendable data format, and it can be manipulated easily in JavaScript. Text file format is the simplest format and can be easily processed by most programming language. In the real practice, JSON is only used in store the

data of users' information and looks' information. For person list and look list, the ID number was directly stored in the Text file line by line. In the Figure 19, there are samples of each four files (Person Data, Look Data, Person List and Look List).

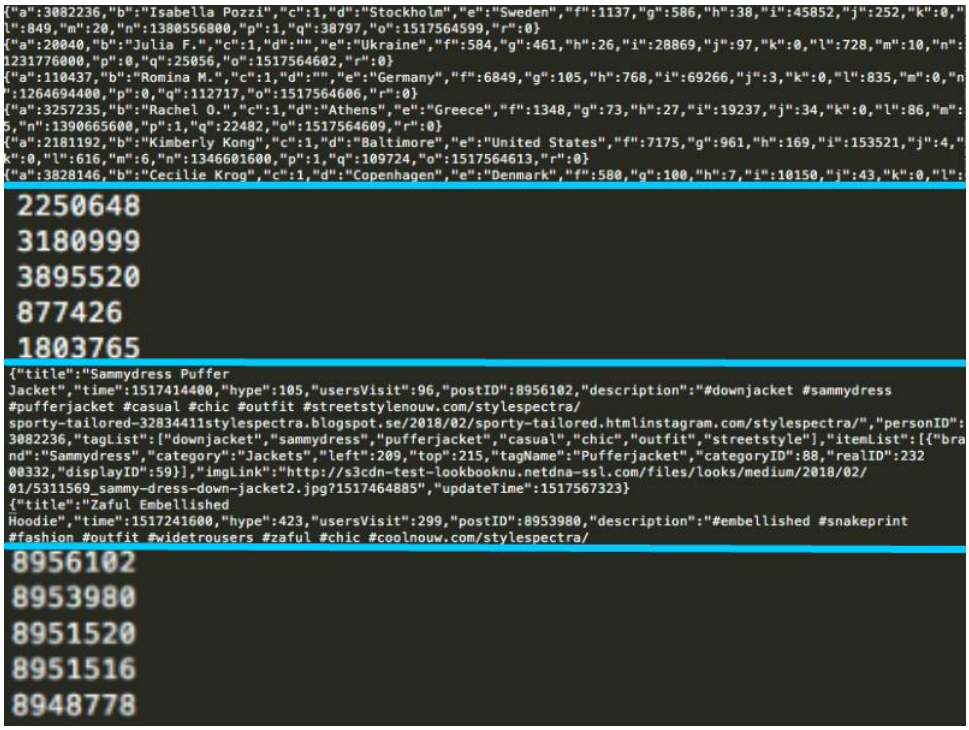

Figure 19: Samples of four files (top to bottom: Person Data, Person List, Look Data and Post List)

4.1.5 First Acceleration Method

Under the basic idea of fetching strategy, the person data was fetched before the look, thus, a more efficient and effective fetching strategy on looks data were formulated by analyzing the person data. A common concept of social network is there are many inactive users who only watch what other active users post and do not post anything on the website. These kinds of users contribute little to the analysis and training, and the best way to deal with them is not fetch them until fetching the active users are completed and classified. It is necessary to give priority to the more useful data since the data quantity is large, and the collection period is time consuming for this project. To determine active and inactive data, a simple analysis was performed to roughly classify the active and inactive users. As previously indicated, the data content of person contains the number of fans and looks of users. If a user has lots of fans and posts lots of images, this user is determined as active. Based on this simple concept, the measurement of the number of looks per person for different fans level is used to solve the problem. To more clearly explain this idea, Figure 20 shows the tendency of this

measurement.

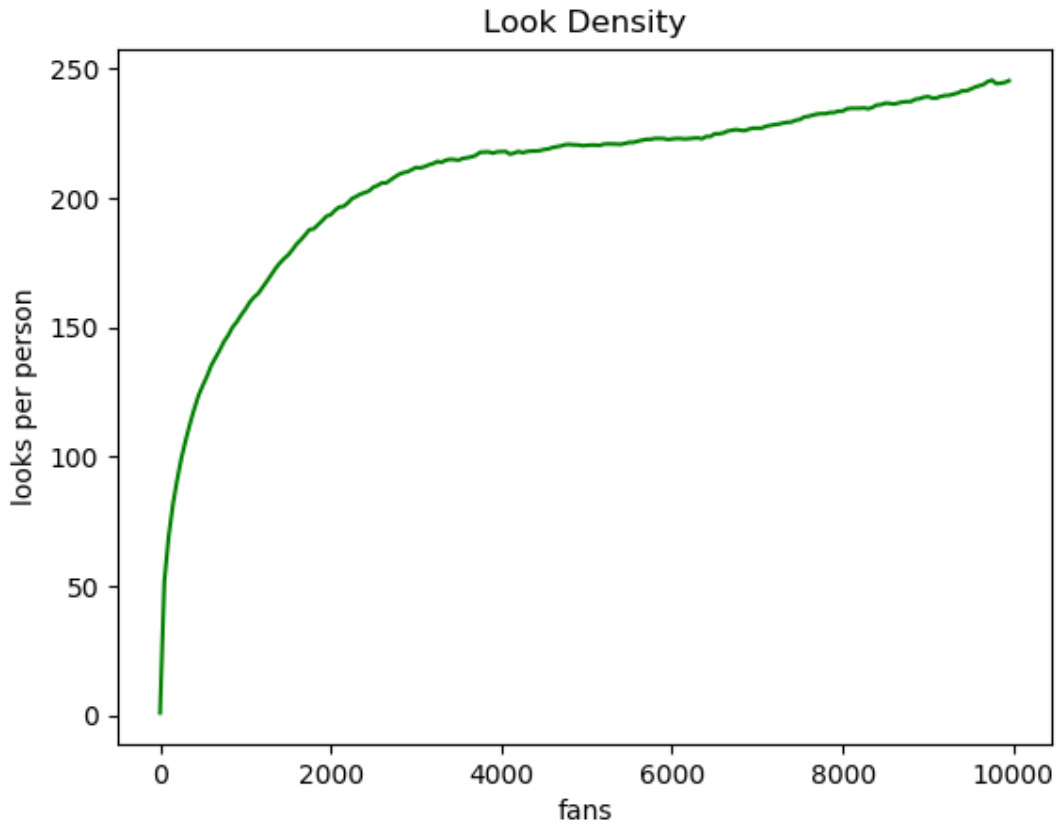

Figure 20: Look density alone fans level

In the Figure 20, the y-axis is the value of looks per person and the x-axis is the number of fans threshold. Each value of looks per person is calculated by the following formula:

$$L = TL/PN$$

Where L is the value of looks per person, TL is the total number of the looks owned by the people, PN is the number of people under some constrain.

In Figure 20, the PN is constrained by the number of fans meaning the number of people who owns fans greater or equal to some value. For example, when the fans threshold is 2,000, the looks per person is calculated from the total number of looks which are owned by the users whose fans number is greater or equal to 2,000 divided by the total number of these users whose fans number is greater or equal to 2,000. This graph indicates that the users who own more fans posted more looks, and the more fans a user owns, the more active the user. As a

result, the users are divided into four groups according to the fans level, which are illustrated in Table 8.

Table 8: Segmented groups

Fans Range	The Number of Users	The Number of Looks	Looks Per Users	Priority (0 lowest to 3 highest)
0 ~ 50	2,510,284	1,388,496	0.5531	0
50 ~ 500	19,727	663,133	33.6155	1
500 ~ 5000	3,787	398,602	105.2553	2
>=5000	911	200,845	220.4665	3

These four groups has fetching priority from high to low, with the group with higher priority should be fetched before the lower one. This strategy ensures the prediction model may start training first with the highest quality data.

4.1.6 Second Acceleration Method

Two Raspberry Pi devices were used simultaneously to speed up data collection. There were some device issues when employed for this purpose. As indicated, limited storage is one of the major issues when using a Raspberry Pi, and was not suitable for fetching photo of the fashion look. Another potential issue was using multiple Raspberry Pi devices simultaneously may cause blocking by the website. Clearly, more Raspberry Pi devices used increased acceleration, but blocking can occur more often. The reason is that multiple Raspberry Pi devices communicated with the fashion website through one Wi-Fi connection using the same public IP address. Therefore, the website may block the public IP of this Wi-Fi since there may be too many requests from the same IP address. Using multiple Raspberry Pi devices may shut down the web server directly by sending a huge number of requests at the same time.

In practice, using only two Raspberry Pi devices plus one laptop were suitable for balancing the speed and the stability of the data collection. The Raspberry Pi devices were only responsible to execute the collection job for the first two crawlers due to its limited storage.

4.2 Data Management and Analysis

4.2.1 Basic Data Statistic

Up until October 21, 2017, the crawler collected 2,534,709 records of users and 1,109,909 of looks. The collection of the first three groups was completed, and the basic data statistic is presented in Tables 9 and 10. Among the data, some of images were broken during collection. Therefore, 1,109,182 integrated data were picked from the whole set and randomly separated into tree subsets for training, validating and testing. The number of data records for the indicated purposes were 800,000, 200,000, and 109,182 respectively.

Table 9: Basic statistic of the users in first three groups

Attribute	Std.	Avg.	1st Quartile	Median	3st Quartile	Max	Min
Fans	5148.23	1045.26	74	134	354	203438	50
Looks	84.44	51.69	10	25	58	1962	0
Heart	293.10	44.92	2	6	20	20355	0
Karma	29673.09	6836.33	540	1381	3790	964167	0
Following	1170.47	229.89	22	61	163	120668	0
Topics	2.54	0.23	1	1	1	214	0
Comment	4828.81	802.49	30	130	470	313853	0
CKarma	239.81	21.14	0	0	2	14450	0
Views	58158.01	14620.08	3537	5809	10863	2825501	11

Table 10: Basic statistic of the looks in first three groups

Attribute	Std.	Avg.	1st Quartile	Median	3st Quartile	Max	Min
Hype	262.12	147.28	29	70	150	9851	0
UserVisit	3224.43	1619.40	469	812	1498	302293	6
TagNum	6.40	2.33	1	1	1	298	0
ItemNum	1.98	2.34	1	2	4	61	0

In the above two tables, 12 attributes were chosen for basic statistical analysis. The geolocation data (city, country), text data (name, title), time data (since, postTime, updateTime, updateDate), ID data (user ID, LookID, TagID, Item ID), and binary data (gender, position) were not counted for different reasons. For geolocation data, a text format was used instead of longitude and latitude or some other numerical value, and cannot be processed into a suitable format for input of MLP. Text data cannot be processed directly since it is not a numerical format. The format of time data was timestamp and it was treated differently and will be introduced later. ID data was useful for management but meaningless for training, thus it was ignored. Finally, binary data which only holds 0 or 1 was unnecessary to perform the statistic.

In the data statistic, almost all standard deviations of attributes were large except ItemNum, TagNum and Topics, which meant the data is decentralized. By comparing the max value with the average value, it was clear that outlier data must exist among the data.

4.2.2 Normalized Data Statistic and Distribution

Using Data normalization to train the network one by one is one option, but it wastes time. Therefore, some other analysis was necessary to reduce risk. Different normalization methods work best for different data distributions; therefore the analytical focus was on the data distribution itself. To analyze normalization methods to process the data, the hype attribute

was selected as an example. Hype is important since this is the ground truth of the output of the MLP. The distribution graph of the Linear hype data is shown Figure 21, and the dense part of the distribution holds a very small value compared with the highest data value. This kind of dataset may still be used in the training directly, but the problem is how to measure the performance of system during or after training. The data mainly reside in a small value. The model needs to output the relatively small value to be valid. The reason why this situation may occur is if the final measurement of the performance of the system is the average difference between the output of the system and the ground truth, this value is still small because most of the data is small, and bigger data cannot affect the measurement lot since the data quantity is large. And it also violates the idea that "the popular things are the most common things", which means the fashion looks which hold thousands of hypes are absolutely good but the fashion looks which hold the most common hype values are not bad. As a result, the normalization method which is suitable for this project was a method which may make the data distribution less extreme. For the great majority of the looks, they should deem the looks as good.

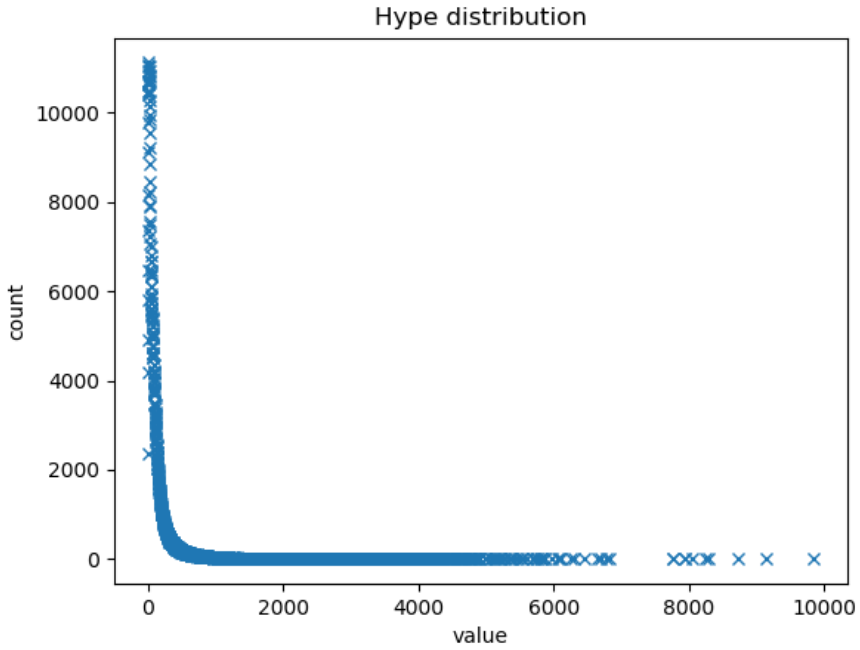

Figure 21: Distribution of raw hype data

In Figure 22, the hype data distributions after normalizations by Linear, Log, Arctan and Sigmoid functions are shown.

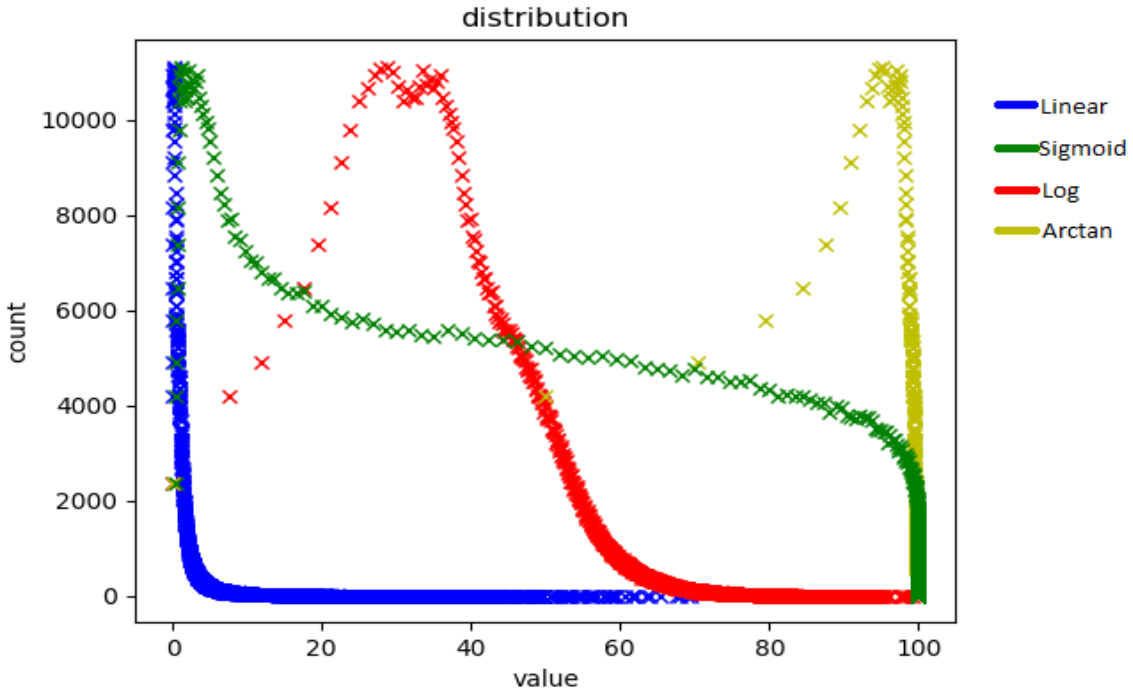

Figure 22: Distributions of normalized hype data

As displayed in Figure 22, the Linear function does not reshape the distribution curve of the hype data. Arctan function reshapes the data into an extreme situation on the opposite side, where most of the data gathered reside in the high mark part. The data distribution processed by Log function and Sigmoid function looks better; the two functions distributed the data relatively evenly in every part from 0 to 1, where the value range from 0 to 1 was scaled to range 0 to 100 to make the graph more readable. But the Log function was closer to the normal distribution. As a result, the Log function was chosen as the normalization method to rescale the data. Other attributes of data were normalized using the same method.

4.2.3 Data Normalization after Removing Top Part

To reduce the effect of outlier values, the top part of data were removed from the normalization process and the data dealt directly as the max value. In practice, the hype value at the position of Top 5% and Top 1% is shown in Table 11. The value of at the Top 1% or 5% position is the max value used in the Log normalization with the top part removed. Data greater than this value were all dealt as max value. The Log normalization method used this value as the max value as well.

Table 11: Value of attributes at Top 1% and Top 5%

Attributes	Value at Top 1%	Value at Top 5%
Hype	1320	550
UserVisit	15400	5540
TagNum	28	11
ItemNum	7	5
Fans	21100	3000
Looks	404	187
Heart	675	142
Karma	98500	24200
Following	2970	792
Topics	4	0
Comment	10500	2700
CKarma	380	40
Views	162000	42000

Since the normalization method used was Log, to avoid the value of Log equal to zero, the number one was added to each value during normalization. Beside the attributes shown in Table 11, there are data attributes related to time. As discussed in Chapter 3.2.2, the max value of time after processing was 114. To keep data at same levels, all time related data were divided by 100.

Top 5% and Top 1% data removals were performed and the distribution of Hype data is shown in Figure 23.

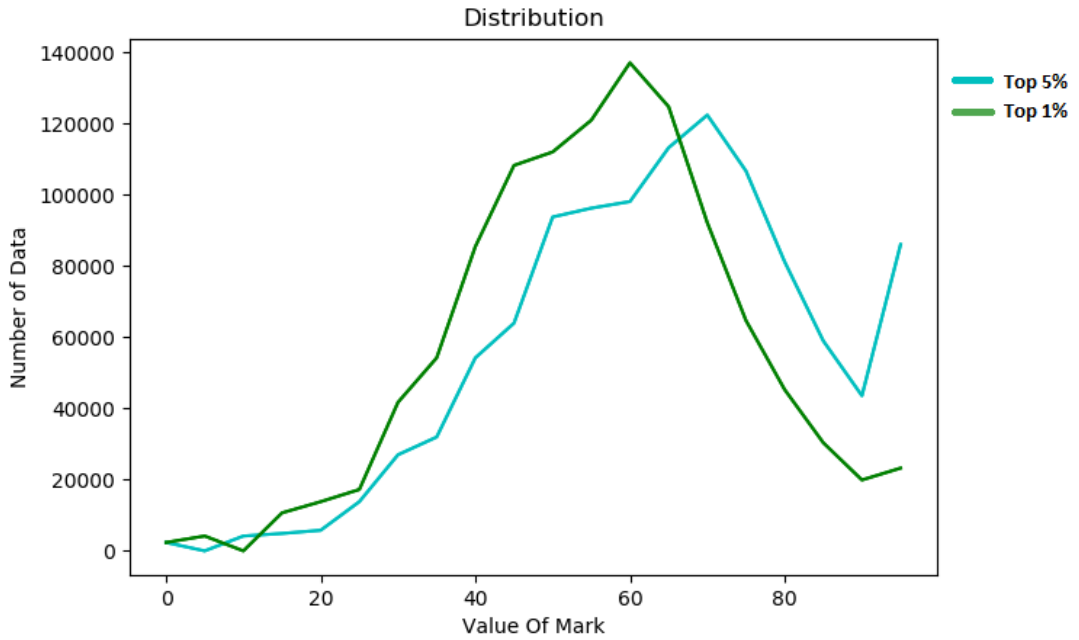

Figure 23: Distribution of normalized hype data after removing top 5% and 1%

As shown in the above figure, the distribution after removing Top 1% was more centralized than Top 5%. Data distribution after removing Top 1% is closer to the normal distribution, which holds a good statistical feature, than the data distribution after removing Top 5%. Therefore, the Top 1% removing plan was used for the data processing method.

4.3 Prediction Model and Model Training

4.3.1 Training Machines

For this project, two machines were used for training. One machine was an Apple Mac Pro with CPU 3.7 GHz Quad-Core Intel Xeon E5, while the other was a Windows computer with CPU 3.40GHz Intel Core i7-6700 and NVIDIA GTX 1070 GPU. The Mac computer has a

better CPU but does not have a suitable GPU, while the Windows computer has a relatively poor CPU but equipped with a powerful GPU. Since neural network training was 10 or more times faster by employing a NVIDA GPU than a normal CPU in Tensorflow, the training was mainly conducted on the Windows machine although the Mac can still be used to assist training. Normally, training takes place over several days or even weeks with one training session conducted each time. During training, some small experiments were conducted on the Mac computer. Compiling during from source code improved the speed of the Tensorflow and was easily accomplished using the Mac OS.

4.3.2 Model Code and Training Experiment of Network 1

The model was constructed by the using the Python API of Tensorflow, and is shown in Figures 36 and 37 in Appendix C. The code redraws the structure of CPLs and MLP of Xception discussed in Chapter 3.3.3. It mapped the last layer of MLP to the 16 fashion landmark coordinates and 16 fashion landmark visibilities.

Network 1 was trained using the dataset of DeepFashion which contained the data of the full body. But in Network 1, only the landmark part was used. There were a total of 65,506, 12,532, and 12,512 data records in the training, validation and test sets respectively. As mentioned in the original paper, the loss function included two parts: landmark and visibility. The loss function of landmark used in this project was the same as indicated in the original paper. The loss function of visibility was Softmax with Cross Entropy. The prediction job of each landmark's visibility was two classes classification. During the training, the batch size was 10 and the optimization algorithm was the Adam Optimizer [49] with the default parameters and the initial learning rate 0.0001. The training took 448500 steps (about 69 epochs). Dropout with the 0.5 dropout rate was used in the final 2 MLP layers. Figure 24 shows the loss curve of the training. This curve is the value of the loss function on the validation dataset. Since this model trained in the early stage of this project, there was no loss record on the training set.

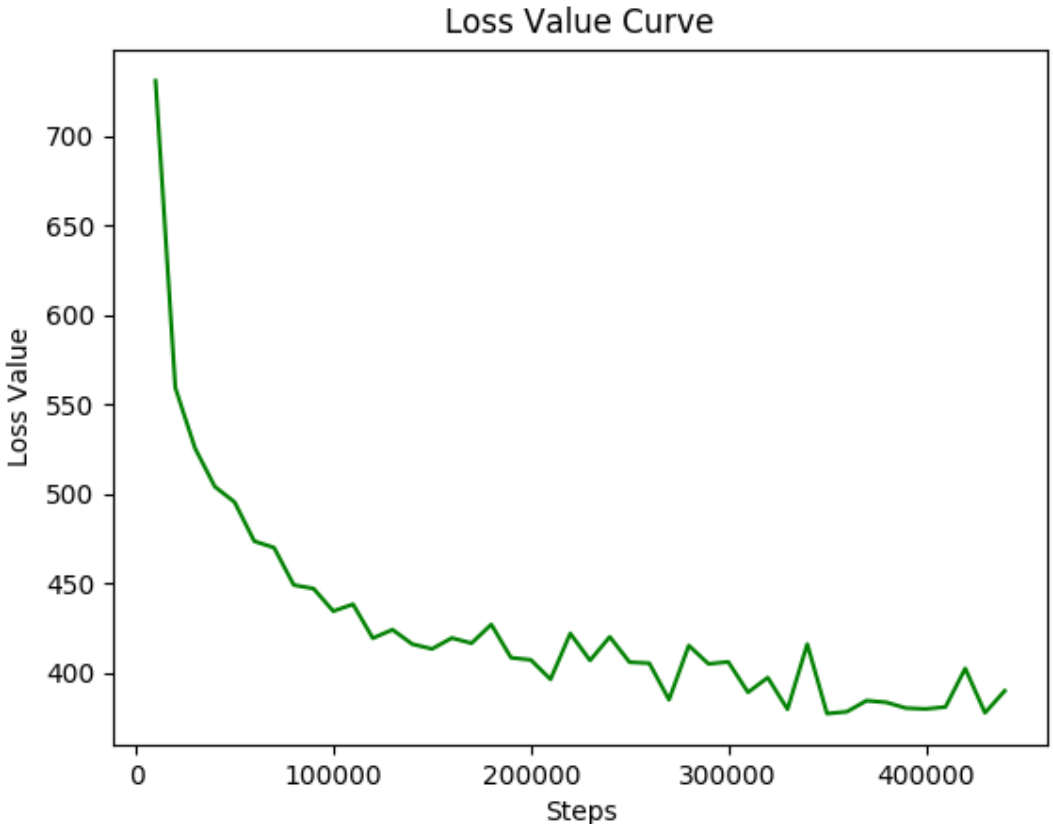

Figure 24: Loss curve of the Network 1 training

The model slightly outperformed the result of landmark detection rate on the test set compared with the results from original paper. The detection rate was measured by Percentage of Detected Joints (PDJ) [6] which was same as the original paper. The detailed performance are shown in Chapter 5.

4.3.3 Model Code and Training Experiment of Network 2

Network 2 has the same network structure as Network 1 but performed more jobs on the output. The classification jobs on the category and attributes of the fashion were extra jobs compared with Network 1. Among the two new jobs, the job related to category was a single-label classification problem which meant each fashion only can be classified into one and only one category. The attributes classification was a multi-label job which indicates one fashion may hold more than one attribute. As a result, the loss function for category classification was Softmax Cross Entropy and the attributes classification was the Sigmoid Cross Entropy. These two loss functions are combined with the loss function in Network 1 as the final loss function of the Network 2. However, to make training processing faster, weight parameters with value 5 was put on the loss function in Network 1 to enhance the

performance on the landmark prediction. The major code of network architecture is shown in Figures 38 and 39 in Appendix C.

The training batch size was 8 with the default Adam Optimizer and 0.001 initial learning rate. No Dropout was added in the network. Instead, Batch Normalization was added between each layer. The decay value and epsilon value of the Batch Normalization was 0.95 and 0.00001 respectively. The accuracy of landmark was still measured by PDJ. Additionally, the performance of classification on categories and attributes was measured by Top-1 and Top-5 Accuracy which was the same as the original paper. Top-N Accuracy meant that the predication was correct only if any of the N highest probability output of the model matched the expected answer. The loss curve is displayed in Figure 25. The blue and green curves are the loss of training and validation respectively. The training process continued for 256,000 (about 32 epochs) steps. The overfitting occurred lightly in the training. The performance achieved, approximately the same levels as in the results from the original paper, is shown in Chapter 5.

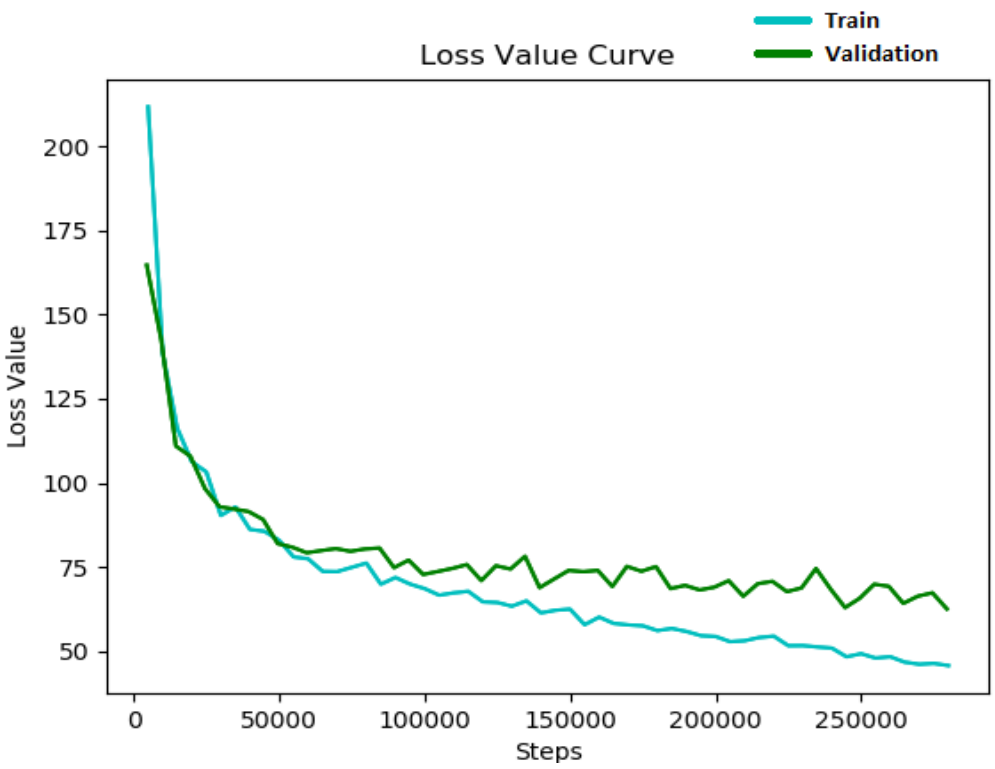

Figure 25: Loss curve of the Network 2 training

4.3.4 Model Code and Training Experiment of Network 3

Starting from the Network 3, the network is responsible for the job of fashion mark regression. Network 3 was the MLP part of the Plan 1 - it used the output of the CPLs of Network 1 with social network supplementary data to perform prediction of the fashion mark. The coding for building Network 3 in the Tensorflow is shown in Figure 26. It has a two layer MLP with 1024 hidden units in each layer. The output is only 1 value, which is the fashion landmark. The final output was rescaled using Sigmoid activation.

```
def MLP(input_x,ist):

    W_MLP1 = weight_variable([25088+18, 1024],'MLP_layer1')
    BN1=tf.contrib.layers.batch_norm(tf.matmul(input_x, W_MLP1),is_training=ist,fused=True,epsilon=1e-5,scale=True,decay=0.99)
    h_MLP1 = active_function(BN1)

    keep_prob_MLP = tf.placeholder(tf.float32)#0.5
    h_fc_drop_MLP1 = tf.nn.dropout(h_MLP1, keep_prob_MLP)

    W_MLP2  = weight_variable([1024, 1024],'MLP_layer2')
    BN2=tf.contrib.layers.batch_norm(tf.matmul(h_fc_drop_MLP1, W_MLP2),is_training=ist,fused=True,epsilon=1e-5,scale=True,decay=0.99)
    h_MLP2 = active_function(BN2)

    h_fc_drop_MLP2 = tf.nn.dropout(h_MLP2, keep_prob_MLP)

    W_mark = weight_variable([1024, 1],'mark_layer')#x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6
    BN3=tf.contrib.layers.batch_norm(tf.matmul(h_fc_drop_MLP2, W_mark),is_training=ist,fused=True,epsilon=1e-5,scale=True,decay=0.99)
    final_mark = tf.nn.sigmoid(BN3)

    return final_mark,keep_prob_MLP
```

Figure 26: Major code of Network 3

To speed up training and avoid redundant calculation, all fashion photos are first transferred to the feature maps of Network 1 and stored in the hard disk. Network 1 directly worked on the feature maps of the fashion photos to save time. The loss function was Mean Absolute Error (MAE). Dropout with 0.5 dropout rate was added to the two layers. Batch Normalization was added with the decay rate 0.99 and epsilon 0.00001. The training used the Adam Optimizer with 0.001 initial learning rate, default parameters setting and 16 batch size. All data were shuffled before the training as well as at the end of each epoch. The training took 700,000 steps (14 epochs), with the loss curve displayed in Figure 27. The blue curve is the loss of the training dataset, while the green curve is the loss of validation dataset. Since there was a big gap between the training loss and the validation loss, Network 3 indicated an over fitting problem.

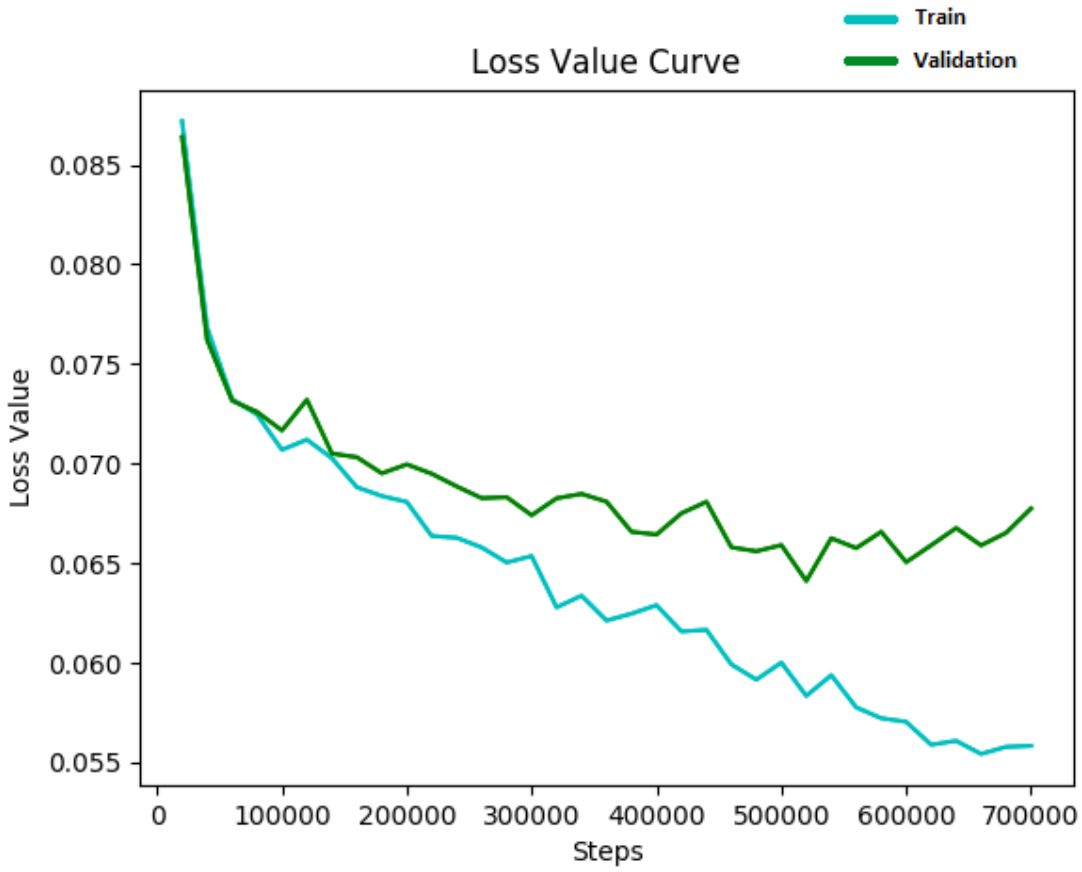

Figure 27: Loss curve of the Network 3 training

However, since the 0.5 dropout rate and Batch Normalization was added to system to overcome the overfitting problem, Plan 1, Network 1 plus Network 3, has little chance to improve performance. The final performance is discussed in the Chapter 5.

4.3.5 Model Code and Training Experiment of Network 4

Networks 4 utilized the feature maps produced by the CPLs part of Network 2 with social network supplementary data to perform the prediction of the fashion mark. The major coding of the network is shown in the Figure 28. MLP contained 2 layers with 1024 hidden units in each layer. The final output which is the mark of fashion was rescaled using Sigmoid activation.

```python
def MLP(input_x,ist):

    W_MLP1 = weight_variable([25088+18, 1024],'MLP_layer1')
    BN1=tf.contrib.layers.batch_norm(tf.matmul(input_x, W_MLP1),is_training=ist,fused=True,epsilon=1e-5,scale=True,decay=0.99)
    h_MLP1 = active_function(BN1)
    #h_MLP1 = tf.nn.leaky_relu(tf.matmul(combined_layer, W_MLP1) + b_MLP1)

    keep_prob_MLP = tf.placeholder(tf.float32)#0.5
    h_fc_drop_MLP1 = tf.nn.dropout(h_MLP1, keep_prob_MLP)

    W_MLP2  = weight_variable([1024, 1024],'MLP_layer2')
    BN2=tf.contrib.layers.batch_norm(tf.matmul(h_fc_drop_MLP1, W_MLP2),is_training=ist,fused=True,epsilon=1e-5,scale=True,decay=0.99)
    h_MLP2 = active_function(BN2)

    h_fc_drop_MLP2 = tf.nn.dropout(h_MLP2, keep_prob_MLP)

    W_mark = weight_variable([1024, 1],'mark_layer')#x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6
    BN3=tf.contrib.layers.batch_norm(tf.matmul(h_fc_drop_MLP2, W_mark),is_training=ist,fused=True,epsilon=1e-5,scale=True,decay=0.99)
    final_mark = tf.nn.sigmoid(BN3)

    return final_mark,keep_prob_MLP
```

Figure 28: Major code of Network 4

As in the training of Network 3, feature maps were produced and stored on the hard disk before training to save time. Loss Function was MAE and the optimizer was Adam with the initial learning rate 0.001, the default parameters setting and 16 batch size. Dropout with dropout rate 0.5 was added for the two layers. Batch Normalization was employed with decay rate 0.99 and epsilon 0.00001. All data were shuffled before training as well as at the end of each epoch. Whole training took 1,500,000 steps (30 epochs) and the two loss curves, training (blue) and validation (green) are displayed in Figure 29. The loss on validation was hard to decrease and overfitting appeared in this model. It was difficult to improve performance by training the network with more steps.

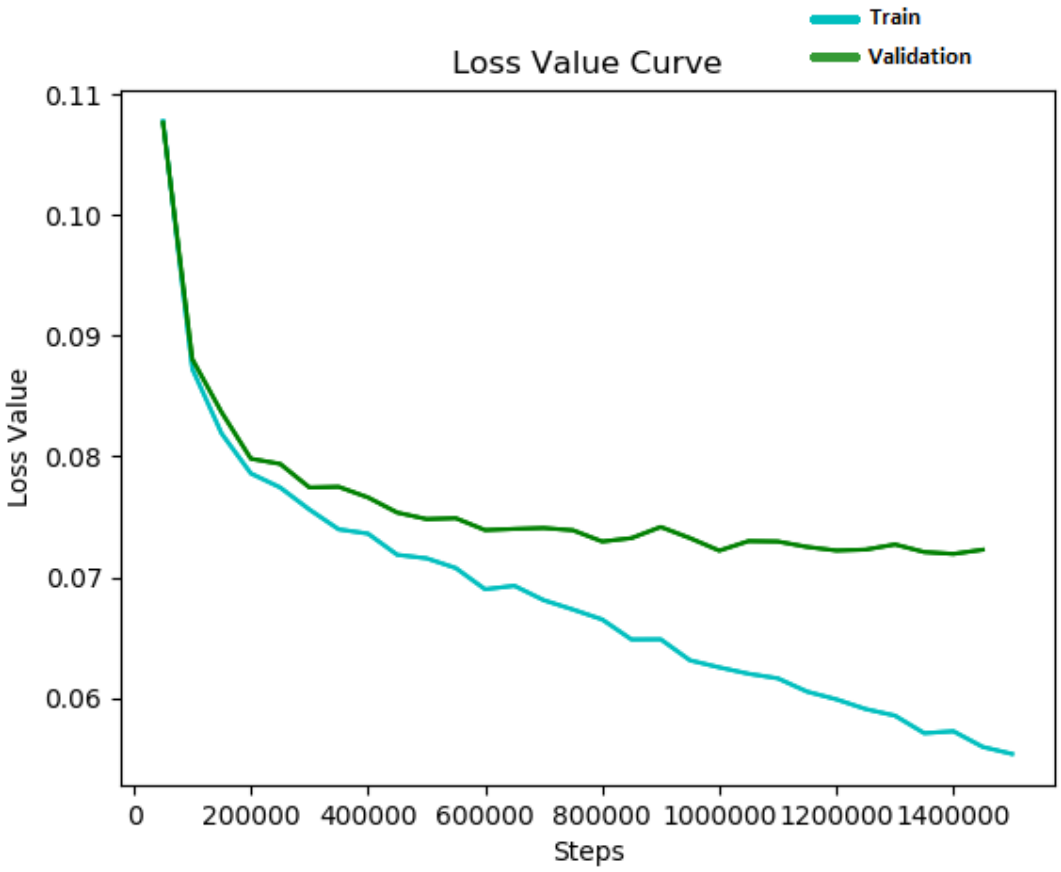

Figure 29: Loss curve of the Network 4 training

4.3.6 Model Code and Training Experiment of Network 5

Network 5 used the outputs of Network 2, but it utilized the final output instead of the output from CPLs part. This is readable to humans and more abstract than the feature maps of the CPLs part. Though it is more abstract, it losses some information compared with the feature maps. The readable data contains a 1000-dimensions vector of attributes, 50-dimensions vector of categories, sixteen values of landmark, and eight 2-dimensions vectors of landmark visibility. Except for the landmark, every value in other vectors data stands for probability of a specific class. Therefore, this vector data was rescaled into the probability format, with a value between 0 and 1, before use for training. The category and visibility data were rescaled by Softmax and attributes data was rescaled by Sigmoid. The network structure is the same as Networks 3 and 4; the code is shown in Figure 30.

```python
def MLP(input_x,ist):

    W_MLP1 = weight_variable([1082+18, 1024],'MLP_layer1')
    BN1=tf.contrib.layers.batch_norm(tf.matmul(input_x, W_MLP1),is_training=ist,fused=True,epsilon=1e-5,scale=True,decay=0.99)
    h_MLP1 = active_function(BN1)
    #h_MLP1 = tf.nn.leaky_relu(tf.matmul(combined_layer, W_MLP1) + b_MLP1)

    keep_prob_MLP = tf.placeholder(tf.float32)#0.5
    h_fc_drop_MLP1 = tf.nn.dropout(h_MLP1, keep_prob_MLP)

    W_MLP2  = weight_variable([1024, 1024],'MLP_layer2')
    BN2=tf.contrib.layers.batch_norm(tf.matmul(h_fc_drop_MLP1, W_MLP2),is_training=ist,fused=True,epsilon=1e-5,scale=True,decay=0.99)
    h_MLP2 = active_function(BN2)

    h_fc_drop_MLP2 = tf.nn.dropout(h_MLP2, keep_prob_MLP)

    W_mark = weight_variable([1024, 1],'mark_layer')#x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6
    BN3=tf.contrib.layers.batch_norm(tf.matmul(h_fc_drop_MLP2, W_mark),is_training=ist,fused=True,epsilon=1e-5,scale=True,decay=0.99)
    final_mark = tf.nn.sigmoid(BN3)

    return final_mark,keep_prob_MLP
```

Figure 30: Major code of Network 5

The final outputs of Network 2 were produced and stored in the hard disk to save training time. All data were shuffled before training and at the end of each epoch. MAE was the loss function and the optimizer was Adam with batch size 16, the default parameters setting and 0.001 initial learning rate. Dropout was employed into the 2 layers with dropout rate 0.5. Batch Normalization was applied with the decay rate 0.99 and epsilon 0.00001. All data were shuffled before training and at the end of each epoch. It took 3,100,000 steps (62 epochs) to complete training, and the loss curve is displayed in Figure 31. The green curve indicates the loss curve on the validation dataset, and the blue curve shows loss on the training dataset. As indicated in the figure, the curve decreases smoothly and shows no overfitting. This network was the smallest in the project which also meant it is also the least time consuming network. Since this network trained quickly, it takes the most training steps. Although there was no overfitting after 62 epochs, the loss decreased extremely slowly even on the training set used in later training periods. This network may still have the possibility for performance improvement but will require huge amount of training steps. The final performance of this network beats the performances of Networks 3 and 4.

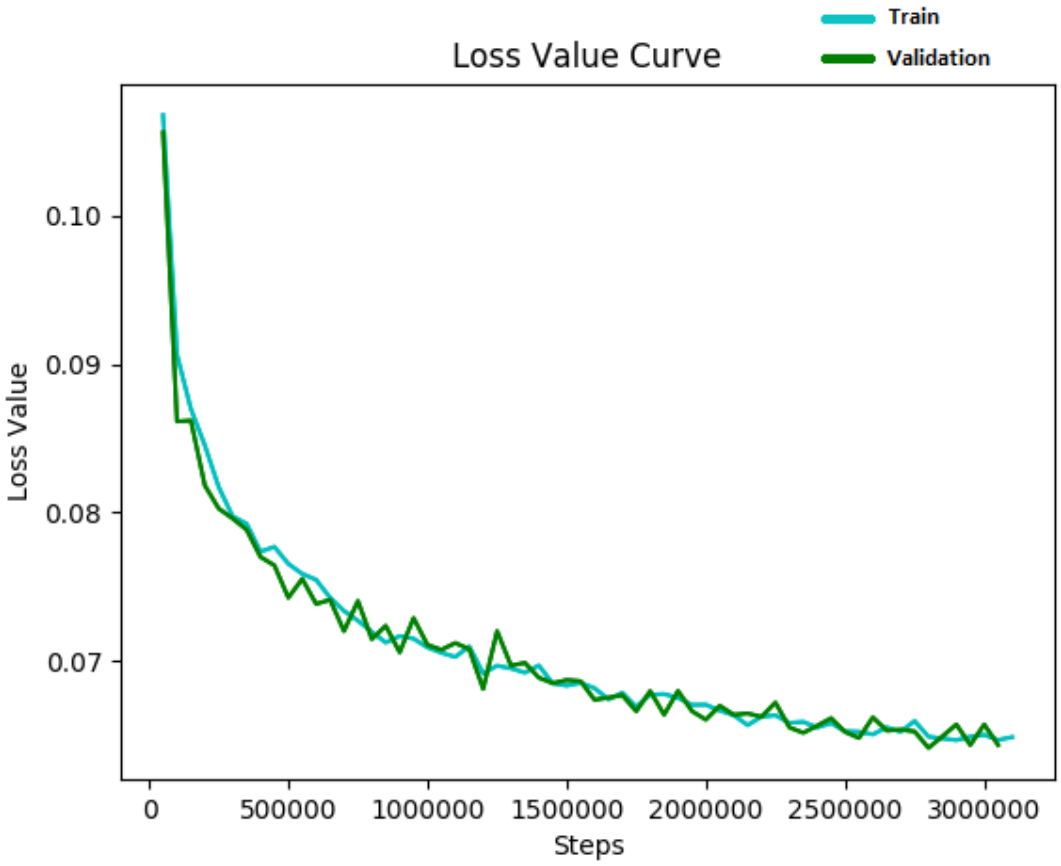

Figure 31: Loss curve of the Network 5 training

4.3.7 Model Code and Training Experiment of Network 6

Network 6 was directly trained using the raw images. This network was very large and the most time consuming. The major coding for Network 6 is shown in Figures 40 to 44 in Appendix C. This code was an implementation of the Xception structure discussed in the Chapter 3.3.3.

Network 6 took the longest time to train, not only because it holds the most complex and huge network structure, but is directly trained using the raw images. The training takes 300,000 steps (12 epochs) and its performance outperformed other networks. The performance detail is discussed in Chapter 5. Dropout with rate 0.5 was applied in the last 2 MLP layers. RMSProp optimizer [50] with default setting, 0.001 initial learning rate, and batch size 16 was utilized to minimize the MAE loss function. Batch Normalization was added between each layer with decay rate 0.99 and epsilon 0.00001. The loss curve is shown in Figure 32.

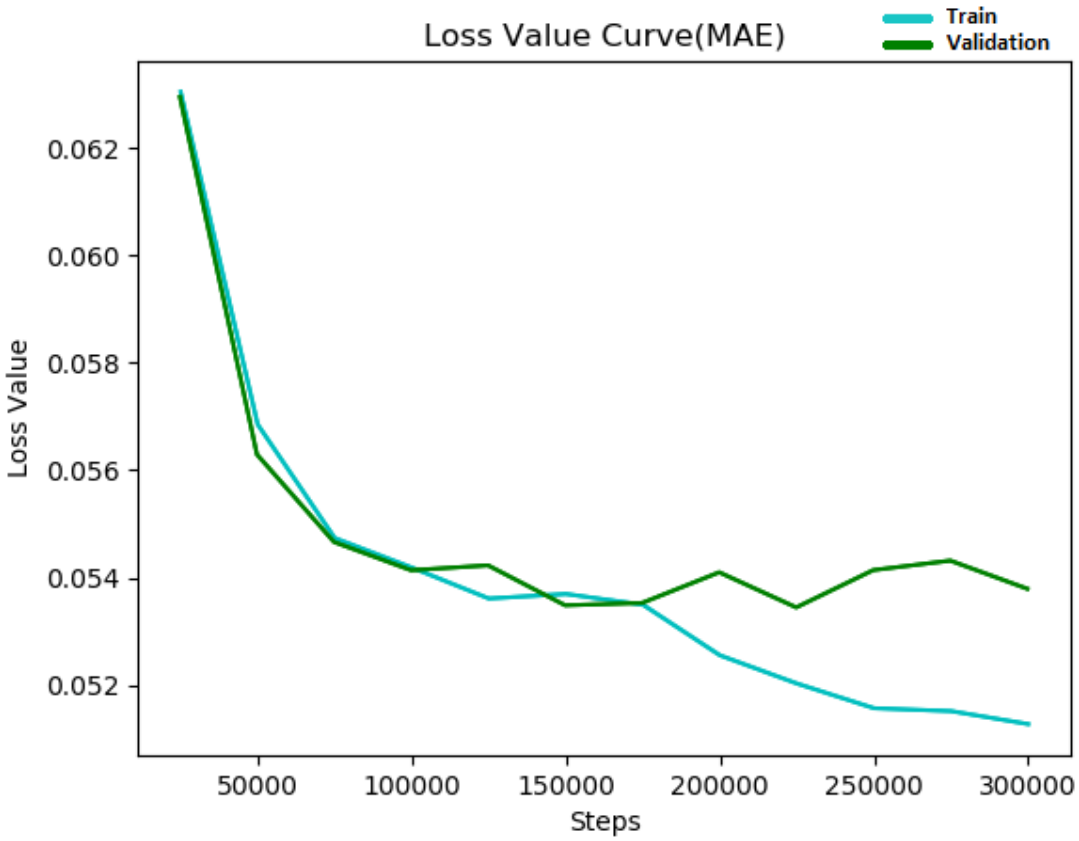

Figure 32: Loss Curve of the Network 6 Training

4.4 Web Application Implementation

The web application was separated into two parts: the front and back-end. Since this was a very simple web application, the workflow was simple as well. First, the front-end send an image to the back. After receiving the image, the back-end processes the image into a suitable format and invokes the neural network model to get the mark of the fashion, then information is returned to the front end. The front-end displays the mark on the webpage.

The front-end was implemented using HTML, CSS and JavaScript files. In order to develop the webpage easily, JQuery [51] was used to control the HTML events. Ajax (Asynchronous JavaScript and XML) played a role of data transmission between the front and back-end.

For the back-end, there were only two routing paths: "/" and "/upload". The first path was for the user to access the webpage. The second path was a POST API for the web page to transfer the image to the server. Figure 33 displays the major coding for the back-end, which used the network of Plan 1. Once the server was launched, the program loaded the neural

network model. When the image is received, the server first saves the image on the local disk and input the image to neural network to get the fashion mark. Since there are 4 plans, the back-end coding to invoke the neural network models were slightly different, but with similar logic.

```
app = Flask(__name__, static_url_path='')
CORS(app)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

@app.route('/')
def hello_world():
    return app.send_static_file('index.html')#render_template('index.html', name='')

@app.route('/upload', methods=['GET','POST'])
def upload_file():
    if request.method == 'POST':
        file = request.files['file']
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            print(filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            imgMatrix=ImgReader(UPLOAD_FOLDER+filename,224,224)

            with CNNs_graph.as_default():
                features=CNNs_sess.run(CNNs_Maps,feed_dict={x_image:imgMatrix})

            with Mark_graph.as_default():
                final_mark=Mark_sess.run(mark_y,feed_dict={feature_x:[np.array(features[0])],supp_x:[supp_data],keep_prob:1.0,ist:False})
            print(final_mark[0][0])
            return str(final_mark[0][0])#redirect(url_for('uploaded_file',filename=filename))


if __name__ == '__main__':
    app.run()
```

Figure 33: Major code of back-end (Plan1)

5 Results and Discussion

5.1 Performance of DeepFashion Model

5.1.1 Performance of Network 1

The performance of Network 1 was tested using a test set provided by the DeepFashion dataset which contained 12,512 data. This test set was never used for training. With the results shown in the Table 12, with the normalized distance of PDJ equal to 0.1, the accuracy of the detection rate achieved 83.40% which is a little higher than the 80% from original paper. There was no specific results mentioned in the original paper when PDJ normalized distance is other than 0.1. Since Network 1 was only responsible for landmark prediction, the result was comparable to the results in the original paper.

Table 12: Performance of Network 1

PDJ Normalized Distance	Accuracy (Ours)	Accuracy (Original paper)
0.10	83.40%	Above 80%
0.08	78.23%	-
0.06	69.40%	-
0.04	52.64%	-
0.02	22.74%	-

5.1.2 Performance of Network 2

The performance of Network 2 was tested on the test set provided by the DeepFashion dataset which contained 12,512 data. Also, this test set was never used for training. The final performance results are shown in the Table 13 indicating performance comparable to the original paper. The performance of PDJ result from the original paper was above 80% when normalized distance was 0.1. There was no specific results mentioned in the original paper when PDJ normalized distance is other than 0.1. Our model performed with 79.51% accuracy on the landmark prediction with the normalized distance of PDJ equaled to 0.1. This result is acceptable for using this model as the Feature Extraction model. For category classification, The Top-3 and Top-5 accuracy of our model was 96.87% and 99.37% respectively which significantly outperformed the results from the original paper. However, in the attribute

classification part, our model did not perform as well as the model from the original paper. The overall performance of our model was acceptable since there was no exhaustive search for hyper parameters.

Table 13: Performance of Network 2

Measurement	Accuracy (Ours)	Accuracy (Original paper)
PDJ (Nor. Distance=0.10)	79.51%	Above 80%
PDJ (Nor. Distance=0.08)	71.82%	-
PDJ (Nor. Distance=0.06)	59.59%	-
PDJ (Nor. Distance=0.04)	39.97%	-
PDJ (Nor. Distance=0.02)	13.89%	-
Category Top-3 Accuracy	96.87%	82.58%
Category Top-5 Accuracy	99.37%	90.17%
Attribute Top-3 Accuracy	34.42%	45.52%
Attribute Top-5 Accuracy	51.02%	54.61%

5.1.3 Summary of the DeepFashion Model

The training of two DeepFashion models was successfully finished. Although the special design structure from original paper was replaced by the single VGG-16 in this project, the performance of these two models demonstrated no big deterioration. Instead, in some jobs, our models even outperformed the results of original paper. The most likely reason was the use of Batch Normalization. There may be other hyper parameters that affected our models performance, but the results were quite good for this project.

5.2 Performance of Fashion Mark Prediction Model

The test results for fashion mark prediction model, Networks 3, 4, 5 and 6, were based on the 200,000 images test set (Data-200K) and 109,182 images validation set (Data-109K) which was never used in the training processes. There was also another test which used 395 new images (Data-395) posted on *lookbook.nu*. The Data-200K and Data-109K were the test and validation set collected before November 2017 with the training set. For the new set Data-395, it was collected on February 1, 2018. Since the actual output of the network is a value from 0 to 1, to make the result more readable, the outputs were multiplied by 100. To measure the

results by MOM and MTPR, the Bin Sizes chosen were 5, 10, 20, 30 and 40 to evaluate performance respectively.

5.2.1 Performance of Network 3

The performance of Network 3 is listed in Table 14. As the Bin Size is increased, the accuracy rules are relaxed. When the Bin Size was above 30, the accuracy exceeded 90% on the two large dataset. When Bin Size equaled 10, the accuracy was around 50%. To some extent, this indicated the model learned the pattern inside the dataset. This model performed better than Network 4 but not as well as Networks 5 and 6.

Table 14: Performance of Network 3

Measurement	Data-200K	Data-109K	Data-395
MOM (Bin Size = 5)	25.80 %	25.83 %	14.94 %
MOM (Bin Size = 10)	49.59 %	49.71 %	35.70 %
MOM (Bin Size = 20)	79.27 %	79.35 %	68.86 %
MOM (Bin Size = 30)	91.66 %	91.81 %	86.84 %
MOM (Bin Size = 40)	96.36 %	96.41 %	95.70 %
MTPR (Bin Size = 5)	20.45 %	20.60 %	14.18 %
MTPR (Bin Size = 10)	37.30 %	37.62 %	26.33 %
MTPR (Bin Size = 20)	61.39 %	61.64 %	46.33 %
MTPR (Bin Size = 30)	77.10 %	77.18 %	61.27 %
MTPR (Bin Size = 40)	86.78 %	87.04 %	71.90 %

5.2.2 Performance of Network 4

Table 15 shows the final performance of Network 4 which was the worst performing model among the 4 fashion mark prediction models. Although it was the worst one, it still achieved around 90% accuracy when Bin Size was set to 30, which was only 2% lower than Network 3 using the Data-200K and Data-109K sets.

Table 15: Performance of Network 4

Measurement	Data-200K	Data-109K	Data-395
MOM (Bin Size = 5)	23.45 %	23.56 %	10.13 %
MOM (Bin Size = 10)	45.84 %	45.73 %	21.01 %
MOM (Bin Size = 20)	75.82 %	75.98 %	43.29 %
MOM (Bin Size = 30)	89.65 %	89.81 %	67.59 %
MOM (Bin Size = 40)	95.52 %	95.50 %	84.30 %
MTPR (Bin Size = 5)	18.56 %	18.52 %	12.41 %
MTPR (Bin Size = 10)	34.27 %	34.31 %	26.58 %
MTPR (Bin Size = 20)	57.85 %	57.90 %	41.77 %
MTPR (Bin Size = 30)	73.62 %	73.70 %	54.94 %
MTPR (Bin Size = 40)	84.02 %	84.20 %	65.06 %

5.2.3 Performance of Network 5

The final performance of Network 5 is listed in Table 16, which showed the good performance among the first three models under current training steps. With Data-200K and Data-109K, there was no improvement when Bin Size was above 30, but accuracy improved by about 6% and 10% compared to Networks 3 and 4 respectively when Bin Size was set to 10.

Table 16: Performance of Network 5

Measurement	Data-200K	Data-109K	Data-395
MOM (Bin Size = 5)	30.61 %	30.42 %	21.27 %
MOM (Bin Size = 10)	55.05 %	55.15 %	42.28 %
MOM (Bin Size = 20)	82.33 %	82.39 %	76.46 %
MOM (Bin Size = 30)	92.58 %	92.76 %	90.38 %
MOM (Bin Size = 40)	96.66 %	96.71 %	98.48 %
MTPR (Bin Size = 5)	23.20 %	23.30 %	13.92 %
MTPR (Bin Size = 10)	41.76 %	41.82 %	24.81 %
MTPR (Bin Size = 20)	66.50 %	66.63 %	43.80 %
MTPR (Bin Size = 30)	81.47 %	81.57 %	57.97 %
MTPR (Bin Size = 40)	89.96 %	90.19 %	71.65 %

5.2.4 Performance of Network 6

As indicated in the Table 17, Network 6 achieved the best performance among the four plans. It significantly outperforms other plans in most of the measurement criteria. This indicates the mixed model may better learn the pattern from the data. Although during the training procedure, it took much longer time than other three plans.

Table 17: Performance of Network 6

Measurement	Data-200K	Data-109K	Data-395
MOM (Bin Size = 5)	32.95 %	32.84 %	21.52 %
MOM (Bin Size = 10)	57.42 %	57.43 %	43.04 %
MOM (Bin Size = 20)	84.38 %	84.43 %	76.71 %
MOM (Bin Size = 30)	93.99 %	94.12 %	92.15 %
MOM (Bin Size = 40)	97.54 %	97.55 %	98.23 %
MTPR (Bin Size = 5)	25.87 %	25.72 %	14.68 %
MTPR (Bin Size = 10)	45.26 %	45.32 %	25.57 %
MTPR (Bin Size = 20)	70.13 %	70.33 %	44.30 %
MTPR (Bin Size = 30)	84.21 %	84.31 %	58.23 %
MTPR (Bin Size = 40)	93.74 %	93.91 %	68.13 %

5.2.5 Summary of the Fashion Mark Prediction Model

Tables 18 and 19 summarizes the comparison between the four fashion mark prediction models using the measurement of MOM and the MTPR with Bin Size equal to 10. There are two reasons to use results using Bin Size equal to 20 for comparison. First, if the Bin Size is large (above 20), the performance differences between the models are not obvious. Second, there is no need to make the measurement too strict because judging the fashion is very subjective. Since this process is difficult even for humans, the model need not be too accurate. On the contrary, it should still perform predictions with accuracy. Since the performance of

these 4 networks is also the performance of 4 plans, the plan numbers are placed in the brackets in the tables.

Table 18: Comparison of Network 3, 4, 5, 6 by MOM (Bin Size = 20)

Measurement	Data-200K	Data-109K	Data-395
Network 3 (Plan 1)	79.27 %	79.35 %	68.86 %
Network 4 (Plan 2)	75.82 %	75.98 %	43.29 %
Network 5 (Plan 3)	82.33 %	82.39 %	76.46 %
Network 6 (Plan 4)	**84.38 %**	**84.42 %**	**76.71 %**

Table 19: Comparison of Network 3, 4, 5, 6 by MTPR (Bin Size = 20)

Measurement	Data-200K	Data-109K	Data-395
Network 3 (Plan 1)	61.39 %	61.14 %	**46.33 %**
Network 4 (Plan 2)	57.85 %	57.90 %	41.77 %
Network 5 (Plan 3)	66.50 %	66.63 %	43.80 %
Network 6 (Plan 4)	**70.15 %**	**70.33 %**	44.30 %

In Tables 18 and 19, the best score belonging to Network 6 (plan4) at around 84% on Data-200k and Data-109K when measured by MOM. The performance has already been suitable for real practice using; and it may still be improved in the future work. Networks 3, 4 and 5 were the separated training model, which utilized the CNN trained on the DeepFashion dataset for feature extraction. Between Networks 3 and 4, both used the output of CPLs as the training input but the CPLs of Network 3 only used the landmarks data without attributes and categories data compared with the CPLs part of Network 4. However, Network 4 strangely performed worse than Network 3 after utilizing more types of data. This situation is may be due to the use of Batch Normalization which affects the output of CPLs of Network 4. Or the attributes and category data provided more noise than the helpful factors in the training of CNNs. There may be other reasons. Additionally, it is obvious that all model performed better on the Data-200K and Data-109K rather than the Data-395. This may be due to the small size of Data-395 which cannot perform a suitable data distribution compared with the

large dataset. The second possibility is because of the time variance of Data-395 and training data. As mentioned in the beginning of Chapter 5.2, Data-395 was collected more than 3 months later than the training set. As the times change, the data distribution was different which affected the model's capacity for predicting future fashion. This is one of the limitations of this project. Comparing the performance of the 4 networks using MOM and MTPR, it is obvious the performance on MOM is better than the MTPR. This indicates the network has more capacity on predicting the correct marking directly rather than predicting a correct order of the whole dataset. Perhaps it is because the loss function (MAE) directly optimized the marking ability of the network rather than output the correct order.

Table 20: Comparison of Network 3, 4, 5, 6 in other parameters

Measurement	Train Epochs	Overfitting	Dropout Rate
Network 3	14	Yes	0.5
Network 4	30	Yes	0.5
Network 5	62	No	0.9
Network 6	12	Yes	0.5

In Table 20, on other aspects of view, a comparison between 4 networks is showed. In terms of the training epochs, Network 5 uses the most steps to optimize the network and it did not experience overfitting. Networks 3, 4 and 6 take relatively fewer steps than Network 5, but they demonstrate overfitting with the dropout rate 0.5. Therefore, it is hard to further reduce the overfitting on Networks 3, 4 and 6. Although Network 5 has not experienced overfitting, the descent is very small which potentially indicates that Network 5 is in a relatively late stage of training. Though it is still possible to improve the performance of Network 5 with more training steps, the improvement may be slight.

In conclusion, the best possible performance among the 4 models is Plan 4 (Network 6). Besides relatively better capability, the Plan 4 trained the model as whole which reduced the complexity of the training process. However, it takes much greater computational resources as well as a much longer training period compared to the other 3 plans. If the model does not require high accuracy or if good training devices are not available, Plans 1, 2 and 3 are better choices.

5.3 Web Application

It is simple to use the web application to get your own mark on a fashion image. Simply upload a fashion image from your computer and clicking the red button, a bar with a percentage value will show on the screen. A higher percentage value stands for a more fashionable image. In Figure 34, a screen capture of this web application is shown.

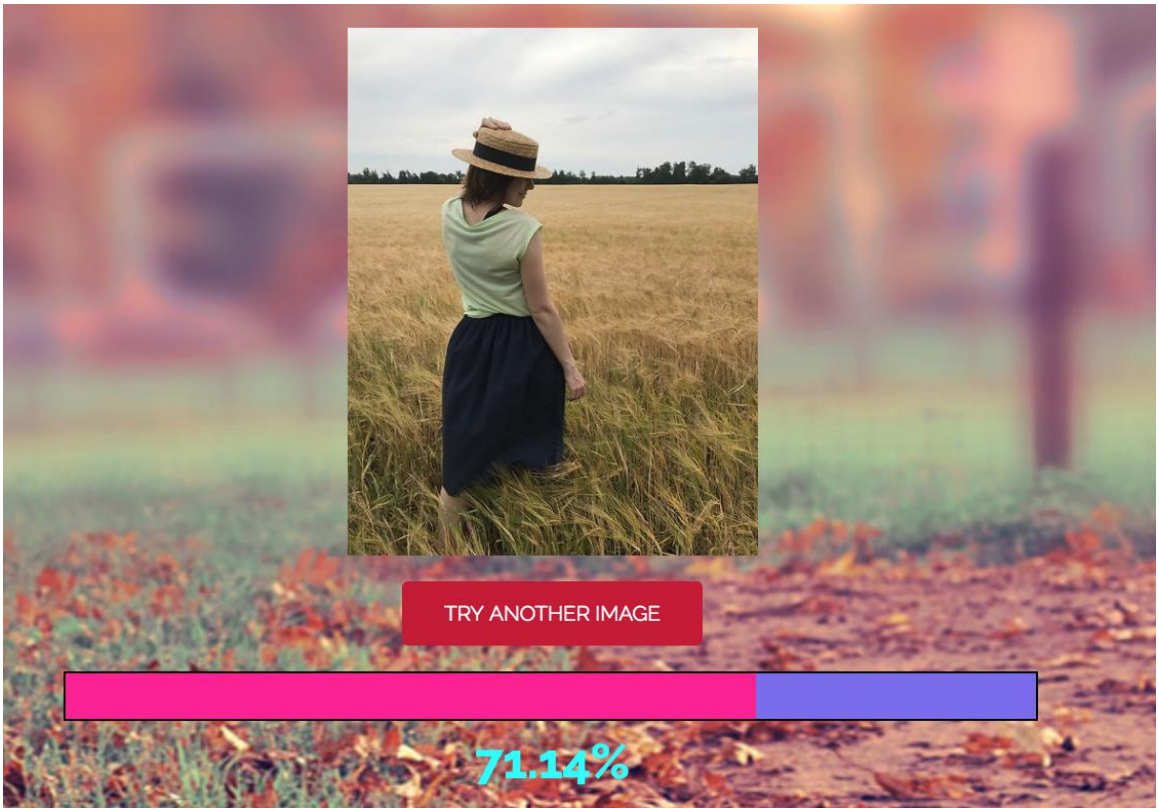

Figure 34: Web application

6 Conclusion and Further Work

6.1 Conclusion

This project presented a completed design and implementation on how to use a deep neural network with data from social network to judge good fashion. The objectives of this project were to collect and process data, then train a model to judge fashion.

Over one million data records (including images and annotation of social network) were collected from the website *lookbook.nu* to construct a large dataset which was used for data analysis, machine learning or other researches. Log data normalization and other pre-processing were performed to improve the quality of the data.

Four plans were proposed and implemented in this project to perform the fashion mark prediction. Plans 1, 2 and 3 utilized a well collected fashion dataset, DeepFashion, to train the VGG-16 CNNs to perform the fashion-related feature extraction. After the feature extraction, the MLP structure was trained to predict the mark of fashion based on the features extracted by VGG-16 trained on DeepFashion. To compare the different feature extractions, the three plans use different sections of dataset to train VGG-16 or used different outputs of the VGG-16. Since the feature extraction models and regression models of these three plans were trained separately, the three plans required relatively low computational resources to be complete its tasks in a reasonable time.

The feature extraction model and regression model of Plan 4 were trained as a whole component. This type of training is more common in the use of modern deep neural networks. Using raw data without any feature extraction, to ensure completeness of the data, for neural network learning is a major advantage of this plan. This plan employed a very advanced network structure, Xception, which brings good performance for the task.

Three test datasets, Data-200K, Data-109K and Data-395 were used to evaluate the performance of the dataset. The results indicate the effectiveness of the four plans. Additionally, a simple web application was designed to apply the deep neural network used in practice.

There are some limitations in this project which should be noted. First, due to the limitation of training devices, it was impossible to perform more searches on better hyper parameters. There is room for improvement for each plan.

Second, data such as text and geolocation were not used in the current models, losing potential ability to judge fashion from other dimensions.

Last, with the use of supplementary data from social networks, the model predicted an accurate mark compared with the real mark value from the social network. This may lead the model to depend on the supplementary data. This may negatively affect the validity of the model without the use of supplementary data.

6.2 Potential Further Work in Future

6.2.1 More Evaluations on the Fashion Mark Prediction Model

The model was successfully trained to predict good fashion. As discussed in Chapter 6, the model performed with acceptable results using a very large test dataset as well as a relatively new small dataset. However, these two evidences only indicate it can perform good results these datasets. More evaluations are needed to further examine the ability of the model. There are two proposed plans to further test the model. The first plan is to utilize new fashion photos from the website. The model can be tested by predicting the future Hype value of a new posted fashion looks. Since the model was trained based on data collected in the past, the model should have the capability to predict images in future. The second plan is to utilize the model to predict Hype values of images produced by our group. These images should be posted on *lookbook.nu* to get the real Hype values and compared with the prediction of the model. These two plans hold similar idea of using the model to predict new fashions.

6.2.2 Model Fine Tune and Model Rebuild

Due to hardware limitations, the training period was affected. Therefore, the results of the current model still have room for improvement. Moreover, this project did not perform exhaustive search on the hyper parameters, which means there is the possibility for improved model training by adjusting the hyper parameters.

Fashion changes all the time. The model cannot be used to predict fashion without constant updates or modifications. Since the model's capability is based on available data, one possible way to maintain or improve performance is to continually update data to fine tune the model. As a result, collection of new data and update frequency are critical issues which need to be considered.

Finally, another possible plan is to rebuild the model by applying new network structure; utilize more types of data, etc. The deep neural network structure establishes the base line performance of the model. It is worthwhile to redesign the model by using much deeper, wider and more powerful network structures. However, since this model maybe embedded in mobile devices which has limited computational resources, the selection of a new structure is situationally dependent. This is discussed in Chapter 6.2.4. The other reason why there may be a need to rebuild the whole model is the addition of new types of data. In this project, the model missed several types of data which may be used to provide better service, such like geographic location and text-based data. The data can be added to the model by modifying the input format of the network.

6.2.3 Reducing Dependence on Social Network Data

In this project, some social network supplementary data were used as input to the MLP part of the model during training. This is a necessary action since the fashion marks are collected from a social network. This means the fashion mark is related to those social networks data. Although training with social network data is a necessary action, it is also one of the major limitations of this project when applying the model into real practical application. Neural networks learned patterns from both social network data and images. However, if the model is used for only image prediction in some applications, the network still needs the input of those social network data which do not exist. In order to use the model, one plan is to make fake supplementary data, and the other plan is to remove or reduce the dependence on those data. Here provides a potential solution of second plan to reduce the model dependence on those social network data: renormalization after training. As what indicated previously, it is necessary to train the model with those data and it is obviously wasteful to redesign and retrain a new version of model without using those supplementary data. Therefore, a renormalization can be done to reduce the model dependency on supplementary data. In renormalization, there still needs to make the fake social network data to input with the image. However, this fake social network data is same for every image in the dataset, which means

every image will be input with one same fake social network data. After inputting the whole dataset, marks for the each record in the dataset will be get. Then this list of mark number will be sorted in order to find the maximum and minimum data. Finally, new maximum and new minimum data will be used to do normalization, like Linear or Log normalization. Those values, which are bigger or smaller than the maximum or minimum, will dealt as the maximum or minimum values. This method can reduce the dependency of model on social network data. The fake social network data will be a hyper parameter which needs experiments for searching the best values.

6.2.4 Mobile-based Application

A mobile-based application may be developed for this project. There are basically two types of mobile-base applications: client-server and local. For a client-server application, it is similar to a web-based application which transmits data from client to the server and relies on the server for computational power. For a local type, the neural network model relies on the mobile device. Since this will require high computational power as well as the development of an API for the device, this may not be a good long term option. A new tool called Tensorflow Lite [52] was recently released which was a specially designed tools for smart phone development. There are some new proposed structures of neural networks specially designed for mobile devices, such as MobileNets [53] and ShuffleNet [54]. These new networks reduce the need for computational resources. There is possibility to redesign a local mobile-based application in the future.

References

[1] Global fashion industry statistics. https://fashionunited.com/global-fashion-industry-statistics#US, [Oct. 26, 2017]

[2] Samuel, Arthur L. Some studies in machine learning using the game of checkers. IBM Journal of research and development, 44.1.2, pages 206-226, 2000

[3] Kingsford, Carl, and S. L. Salzberg. What are decision trees? Nature Biotechnology, 26.9, pages 1011-1013, 2008

[4] Cortes, Corinna, and V. Vapnik. "Support-vector networks." Machine Learning 20.3(1995):273-297

[5] Kriesel, David. A Brief Introduction to Neural Networks. Introduction to Neural Networks. Macmillan Education UK, pages 31-39, 2007.

[6] Liu, Z., Luo, P., Qiu, S., Wang, X., and Tang, X. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. IEEE Conference on Computer Vision and Pattern Recognition, pages 1096-1104, 2016

[7] Liu, Z., Yan, S., Luo, P., Wang, X., and Tang, X. Fashion landmark detection in the wild. In European Conference on Computer Vision, pages 229-245, October 2016. Springer International Publishing.

[8] Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. International Conference on Neural Information Processing Systems, vol.25, pages 1097-1105, 2012. Curran Associates Inc.

[9] Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. Neural Networks, 2(5), pages 359-366, 1989.

[10] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), pages 1929-1958, 2014.

[11] Xiu-Shen Wei. Must Know Tips/Tricks in Deep Neural Networks. http://lamda.nju.edu.cn/weixs/project/CNNTricks/CNNTricks.html, 2015 [Oct. 26, 2017]

[12] WGSN official website. https://www.wgsn.com/cs/, [Oct. 26, 2017]

[13] Nike official website. https://www.nike.com/, [Oct. 26, 2017]

[14] Adidas official website. https://www.adidas.com, [Oct. 26, 2017]

[15] Levis official website. https://www.levi.com, [Oct. 26, 2017]

[16] Coach official website. http://world.coach.com/, [Oct. 26, 2017]

[17] HM official website. http://www.hm.com/, [Oct. 26, 2017]

[18] Clients of WGSN. https://www.wgsn.com/cs/members/#!/page/our-clients, [Oct. 26, 2017]

[19] Prediction product WGSN. https://www.wgsn.com/en/products/fashion/, [Oct. 26, 2017]

[20] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biology, 5(4), pages 115–133, 1943.

[21]Gradient descent lecture notes of UCLA. http://www.math.ucla.edu/~wotaoyin/math273a/slides/Lec3_gradient_descent_273a_2015_f.pdf, [Oct. 26, 2017]

[22]Larasati, Aisyah, C. Deyong, and L. Slevitch. Comparing Neural Network and Ordinal Logistic Regression to Analyze Attitude Responses. Service Science, 3.4, pages 304-312, 2011.

[23] Lecun, Y., et al. Backpropagation applied to handwritten zip code recognition. Neural Computation, 1.4, pages 541-551, 2014.

[24] Simonyan, Karen, and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. Computer Science, 2014.

[25] F. Chollet. Xception: Deep learning with depthwise separable convolutions. arXiv preprint arXiv:1610.02357v2, 2016.

[26] Tensorflow official website. https://www.tensorflow.org/, [Oct. 26, 2017]

[27] Python official website. https://www.python.org/, [Oct. 26, 2017]

[28] NVIDIA official website. http://www.nvidia.com/page/home.html, [Mar. 9, 2018]

[29] Simo-Serra, E., Fidler, S., Moreno-Noguer, F., and Urtasun, R. A High Performance CRF Model for Clothes Parsing. Asian Conference on Computer Vision, vol.9005, pages 64-81, 2014. Springer International Publishing.

[30]Lafferty, J. D., Mccallum, A., and Pereira, F. C. N. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. Eighteenth International Conference on Machine Learning, vol.3, pages 282-289, 2001. Morgan Kaufmann Publishers I.

[31] Simo-Serra, E., and Ishikawa, H. Fashion Style in 128 Floats: Joint Ranking and Classification Using Weak Data for Feature Extraction. IEEE Conference on Computer Vision and Pattern Recognition, pages 298-307, 2016.

[32] Song, Z., Wang, M., Hua, X., and Yan, S. Predicting occupation via human clothing and contexts. 24(4): pages 1084-1091, 2011.

[33] Alhalah, Z., Stiefelhagen, R., and Grauman, K. Fashion forward: forecasting visual style in fashion. 2017

[34] Simo-Serra, E., Fidler, S., Moreno-Noguer, F., and Urtasun, R. Neuroaesthetics in fashion: Modeling the perception of fashionability. IEEE Conference on Computer Vision and Pattern Recognition, pages 869-877, 2015.

[35] Raspberry pi official website. https://www.raspberrypi.org/, [Oct. 26, 2017]

[36] JavaScript official website. https://www.javascript.com/, [Feb. 21, 2018]

[37] Sergey I. and Christian S. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of the 32nd International Conference on Machine Learning, PMLR 37: pages 448-456, 2015.

[38] Clevert, Djork-Arné, Unterthiner T, Hochreiter S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). Computer Science, 2015.

[39] HTML official website. https://www.w3.org/html/, [Feb. 21, 2018]

[40] CSS official website. https://www.w3.org/Style/CSS/Overview.en.html, [Mar. 9, 2018]

[41] Flask official website. http://flask.pocoo.org/, [Mar. 9, 2018]

[42] Node.js official website. https://nodejs.org, [Oct. 26, 2017]

[43] Chrome V8 engine official website. https://developers.google.com/v8/, [Oct. 26, 2017]

[44] Official document of http package in Node.js. https://nodejs.org/api/http.html, [Feb. 21, 2018]

[45] Official document of fs package in Node.js. https://nodejs.org/api/fs.html, [Feb. 21, 2018]

[46] NPM official website for request package. https://www.npmjs.com/package/request, [Feb. 21, 2018]

[47] NPM official website for cheerio package. https://www.npmjs.com/package/cheerio, [Feb. 21, 2018]

[48] JSON official website. http://www.json.org/, [Oct. 26, 2017]

[49] Kingma D, Ba J. Adam: A Method for Stochastic Optimization. Computer Science, 2014.

[50] Geoffrey Hinton, N Srivastava, and Kevin Swersky. Lecture 6a overview of mini–batch gradient descent. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, [Mar. 9, 2017]

[51] JQuery official website. https://jquery.com/, [Mar. 9, 2017]

[52] Tensorflow Lite official website. https://www.tensorflow.org/mobile/tflite/, [Mar. 9, 2017]

[53] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Effi- cient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017

[54] X. Zhang, X. Zhou, L. Mengxiao, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. arXiv preprint arXiv:1707.01083, 2017.

Appendix A. Project Management

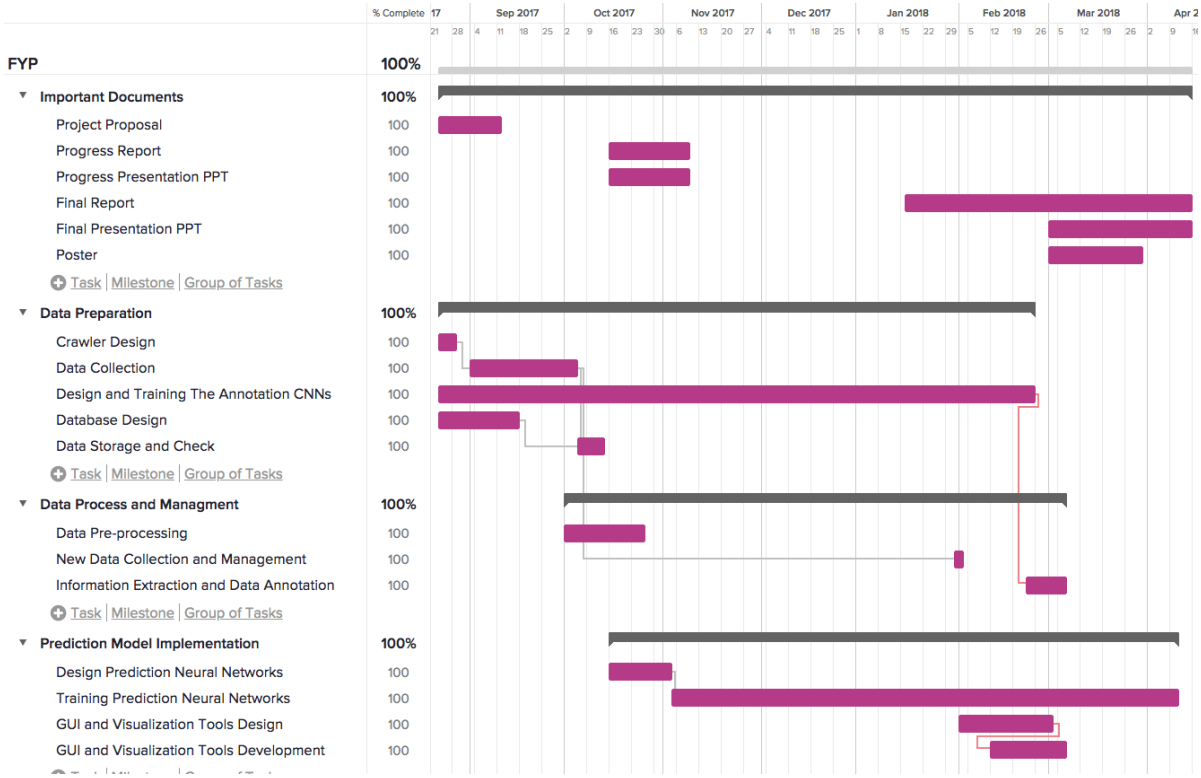

Figure 1: Gantt chart

Appendix B. Reflection

During the whole year of study and work on this project, I do not only learn plenty of theoretical neural network knowledge in detail, but also gain abundant practical experience in neural network engineering jobs. I read lots of papers related to machine learning, neural network, and latest deep learning studies. Additionally, I have been studying Tensorflow APIs for better improving the efficiency and performance of the neural networks. Since the artificial intelligence, more specifically in deep learning, is a quite hot topic recent years and it may also be a very possible factor of next industry revolution, this project gives me an excellent chance to enter this popular area. I believe this will be beneficial in my future career.

In the process of this project, many challengeable problems appeared. Personally speaking, among those barriers, the hardest part of this project is the training process of the neural network, which is also common problem in modern neural network study. Since the neural network holds many hyper parameters which is critical to the performance of the network, how to adjust and find those parameters is very important and time consuming because this needs plenty of experiments. In this project, due to the lack of good GPUs, experiments processed quite slowly which is also the major bottleneck of the project progress. Therefore, I think good hardware plays a critical role in the project which may use large structure of neural networks.

Artificial intelligence is the current trend in the computer science and related areas; this is not revealed in the academic research but also in the practical application. My project is a study of how to put deep learning into an application of specific area. With the growth of the deep learning, I believe that how to apply this powerful theory into real application will be another popular trend in the future. And I think this type of projects will be quite valuable as future topics of final year project.

In summary, this project brings me a full and busy year of my university life. It gives me a chance to study a totally new area and brings me important experience on large project and report writing. I really appreciate this experience.

Appendix C. Program source code / UML diagram, etc.

```
def deepnn(input_x,ist):

  W_conv1 = weight_variable([3, 3, 3, 64],'conv_layer1')#[kernal length, kernal width, input channel, No. of features maps ]
  BN1=batch_norm(conv2d(x, W_conv1),ist)
  h_conv1 = active_function(BN1)#Relu
  # 2st conv layer -- maps 64 feature maps to 64.
  W_conv2 = weight_variable([3, 3, 64, 64],'conv_layer2')
  BN2=batch_norm(conv2d(h_conv1, W_conv2),ist)
  h_conv2 = active_function(BN2)
  # 1st pooling layer.  112*112
  h_pool1 = max_pool_2x2(h_conv2)

  # 3st conv layer -- maps 64 feature maps to 128.
  W_conv3 = weight_variable([3, 3, 64, 128],'conv_layer3')
  BN3=batch_norm(conv2d(h_pool1, W_conv3),ist)
  h_conv3 = active_function(BN3)
  # 4st conv layer -- maps 128 feature maps to 128.
  W_conv4 = weight_variable([3, 3, 128, 128],'conv_layer4')
  BN4=batch_norm(conv2d(h_conv3, W_conv4),ist)
  h_conv4 = active_function(BN4)
  # 2st pooling layer. 56*56
  h_pool2 = max_pool_2x2(h_conv4)

  # 5st conv layer -- maps 128 feature maps to 256.
  W_conv5 = weight_variable([3, 3, 128, 256],'conv_layer5')
  BN5=batch_norm(conv2d(h_pool2, W_conv5),ist)
  h_conv5 = active_function(BN5)
  # 6st conv layer -- maps 256 feature maps to 256.
  W_conv6 = weight_variable([3, 3, 256, 256],'conv_layer6')
  BN6=batch_norm(conv2d(h_conv5, W_conv6),ist)
  h_conv6 = active_function(BN6)
  # 7st conv layer -- maps 256 feature maps to 256.
  W_conv7 = weight_variable([3, 3, 256, 256],'conv_layer7')
  BN7=batch_norm(conv2d(h_conv6, W_conv7),ist)
  h_conv7 = active_function(BN7)
  # 3st pooling layer. 28*28
  h_pool3 = max_pool_2x2(h_conv6)

  # 8st conv layer -- maps 256 feature maps to 512.
  W_conv8 = weight_variable([3, 3, 256, 512],'conv_layer8')
  BN8=batch_norm(conv2d(h_pool3, W_conv8),ist)
  h_conv8 = active_function(BN8)
  # 9st conv layer -- maps 512 feature maps to 512.
  W_conv9 = weight_variable([3, 3, 512, 512],'conv_layer9')
  BN9=batch_norm(conv2d(h_conv8, W_conv9),ist)
  h_conv9 = active_function(BN9)
  # 10st conv layer -- maps 512 feature maps to 512.
  W_conv10 = weight_variable([3, 3, 512, 512],'conv_layer10')
  BN10=batch_norm(conv2d(h_conv9, W_conv10),ist)
  h_conv10 = active_function(BN10)
  # 4st pooling layer. 14*14
  h_pool4 = max_pool_2x2(h_conv10)

  # 11st conv layer -- maps 512 feature maps to 512.
  W_conv11 = weight_variable([3, 3, 512, 512],'conv_layer11')
  BN11=batch_norm(conv2d(h_pool4,W_conv11),ist)
  h_conv11 = active_function(BN11)
  # # 12st conv layer -- maps 512 feature maps to 512.
  W_conv12 = weight_variable([3, 3, 512, 512],'conv_layer12')
  BN12=batch_norm(conv2d(h_conv11, W_conv12),ist)
  h_conv12 = active_function(BN12)
  # # 13st conv layer -- maps 512 feature maps to 512.
  W_conv13 = weight_variable([3, 3, 512, 512],'conv_layer13')
  BN13=batch_norm(conv2d(h_conv12, W_conv13),ist)
  h_conv13 = active_function(BN13)
  # # 5st pooling layer. 7*7
  h_pool5 = max_pool_2x2(h_conv13)
```

Figure 2: Major code of Network 1 (part 1)

```
W_fc1 = weight_variable([7 * 7 * 512, 1024],'MLP_layer1')
h_pool5_flat = tf.reshape(h_pool5, [-1, 7*7*512])
tf.add_to_collection("CNN_MAPs",h_pool5_flat)
h_fc1 = active_function(batch_norm(tf.matmul(h_pool5_flat, W_fc1),ist))

# Dropout - controls the complexity of the model, prevents co-adaptation of
# features.
keep_prob = tf.placeholder(tf.float32)#0.5
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# Fully connected layer 2 --
# 4096 feature to 1024 features.
W_fc2 = weight_variable([1024, 1024],'MLP_layer2')
h_fc2 = active_function(batch_norm(tf.matmul(h_fc1_drop, W_fc2),ist))
h_fc2_drop = tf.nn.dropout(h_fc2, keep_prob)

# Fully connected layer 3 --
# 1024 feature to 18 features.
W_fc_landmark = weight_variable([1024, 16],'lan_layer1')#x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6
b_fc_landmark = bias_variable([16])
fc_landmark = tf.matmul(h_fc2_drop, W_fc_landmark) + b_fc_landmark

W_fc_v1 = weight_variable([1024, 2],'vis_layer1')
b_fc_v1 = bias_variable([2])
fc_v1 = tf.matmul(h_fc2_drop, W_fc_v1) + b_fc_v1

W_fc_v2 = weight_variable([1024, 2],'vis_layer2')
b_fc_v2 = bias_variable([2])
fc_v2 = tf.matmul(h_fc2_drop, W_fc_v2) + b_fc_v2

W_fc_v3 = weight_variable([1024, 2],'vis_layer3')
b_fc_v3 = bias_variable([2])
fc_v3 = tf.matmul(h_fc2_drop, W_fc_v3) + b_fc_v3

W_fc_v4 = weight_variable([1024, 2],'vis_layer4')
b_fc_v4 = bias_variable([2])
fc_v4 = tf.matmul(h_fc2_drop, W_fc_v4) + b_fc_v4

W_fc_v5 = weight_variable([1024, 2],'vis_layer5')
b_fc_v5 = bias_variable([2])
fc_v5 = tf.matmul(h_fc2_drop, W_fc_v5) + b_fc_v5

W_fc_v6 = weight_variable([1024, 2],'vis_layer6')
b_fc_v6 = bias_variable([2])
fc_v6 = tf.matmul(h_fc2_drop, W_fc_v6) + b_fc_v6

W_fc_v7 = weight_variable([1024, 2],'vis_layer7')
b_fc_v7 = bias_variable([2])
fc_v7 = tf.matmul(h_fc2_drop, W_fc_v7) + b_fc_v7

W_fc_v8 = weight_variable([1024, 2],'vis_layer8')
b_fc_v8 = bias_variable([2])
fc_v8 = tf.matmul(h_fc2_drop, W_fc_v8) + b_fc_v8
return fc_landmark, fc_v1, fc_v2, fc_v3, fc_v4, fc_v5, fc_v6, fc_v7, fc_v8,keep_prob
```

Figure 3: Major code of Network 1 (part 2)

```
def deepnn(x,ist):
  #Input Layer 300*300 RGB image [Number of Image, length,width,input channel]

  # 1st conv layer
  W_conv1 = weight_variable([3, 3, 3, 64],'conv_layer1')#[kernal length, kernal width, input channel, No. of features maps ]
  BN1=batch_norm(conv2d(x, W_conv1),ist)
  h_conv1 = active_function(BN1)#Relu
  # 2st conv layer -- maps 64 feature maps to 64.
  W_conv2 = weight_variable([3, 3, 64, 64],'conv_layer2')
  BN2=batch_norm(conv2d(h_conv1, W_conv2),ist)
  h_conv2 = active_function(BN2)
  # 1st pooling layer.  112*112
  h_pool1 = max_pool_2x2(h_conv2)

  # 3st conv layer -- maps 64 feature maps to 128.
  W_conv3 = weight_variable([3, 3, 64, 128],'conv_layer3')
  BN3=batch_norm(conv2d(h_pool1, W_conv3),ist)
  h_conv3 = active_function(BN3)
  # 4st conv layer -- maps 128 feature maps to 128.
  W_conv4 = weight_variable([3, 3, 128, 128],'conv_layer4')
  BN4=batch_norm(conv2d(h_conv3, W_conv4),ist)
  h_conv4 = active_function(BN4)
  # 2st pooling layer. 56*56
  h_pool2 = max_pool_2x2(h_conv4)

  # 5st conv layer -- maps 128 feature maps to 256.
  W_conv5 = weight_variable([3, 3, 128, 256],'conv_layer5')
  BN5=batch_norm(conv2d(h_pool2, W_conv5),ist)
  h_conv5 = active_function(BN5)
  # 6st conv layer -- maps 256 feature maps to 256.
  W_conv6 = weight_variable([3, 3, 256, 256],'conv_layer6')
  BN6=batch_norm(conv2d(h_conv5, W_conv6),ist)
  h_conv6 = active_function(BN6)
  # 7st conv layer -- maps 256 feature maps to 256.
  W_conv7 = weight_variable([3, 3, 256, 256],'conv_layer7')
  BN7=batch_norm(conv2d(h_conv6, W_conv7),ist)
  h_conv7 = active_function(BN7)
  # 3st pooling layer. 28*28
  h_pool3 = max_pool_2x2(h_conv6)

  # 8st conv layer -- maps 256 feature maps to 512.
  W_conv8 = weight_variable([3, 3, 256, 512],'conv_layer8')
  BN8=batch_norm(conv2d(h_pool3, W_conv8),ist)
  h_conv8 = active_function(BN8)
  # 9st conv layer -- maps 512 feature maps to 512.
  W_conv9 = weight_variable([3, 3, 512, 512],'conv_layer9')
  BN9=batch_norm(conv2d(h_conv8, W_conv9),ist)
  h_conv9 = active_function(BN9)
  # 10st conv layer -- maps 512 feature maps to 512.
  W_conv10 = weight_variable([3, 3, 512, 512],'conv_layer10')
  BN10=batch_norm(conv2d(h_conv9, W_conv10),ist)
  h_conv10 = active_function(BN10)
  # 4st pooling layer. 14*14
  h_pool4 = max_pool_2x2(h_conv10)

  # 11st conv layer -- maps 512 feature maps to 512.
  W_conv11 = weight_variable([3, 3, 512, 512],'conv_layer11')
  BN11=batch_norm(conv2d(h_pool4,W_conv11),ist)
  h_conv11 = active_function(BN11)
  # # 12st conv layer -- maps 512 feature maps to 512.
  W_conv12 = weight_variable([3, 3, 512, 512],'conv_layer12')
  BN12=batch_norm(conv2d(h_conv11, W_conv12),ist)
  h_conv12 = active_function(BN12)
  # # 13st conv layer -- maps 512 feature maps to 512.
  W_conv13 = weight_variable([3, 3, 512, 512],'conv_layer13')
  BN13=batch_norm(conv2d(h_conv12, W_conv13),ist)
  h_conv13 = active_function(BN13)
  # # 5st pooling layer. 7*7
  h_pool5 = max_pool_2x2(h_conv13)
```

Figure 4: Major code of Network 2 (part 1)

```
# Fully connected layer 1 --
#after 5 (pooling)round of downsampling, our 224*224 image
# is down to 10x10x512 feature maps -- maps this to 1024 features.
W_fc1 = weight_variable([7 * 7 * 512, 1024],'fc_layer1')
h_pool5_flat = tf.reshape(h_pool5, [-1, 7*7*512])
tf.add_to_collection("CNN_MAPs",h_pool5_flat)
BN14=batch_norm(tf.matmul(h_pool5_flat, W_fc1),ist)
h_fc1 = active_function(BN14)

# features.
keep_prob_MLP = tf.placeholder(tf.float32)#0.5
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob_MLP)

# Fully connected layer 2 --
# 4096 feature to 1024 features.
W_fc2 = weight_variable([1024, 1024],'fc_layer2')
BN15=batch_norm(tf.matmul(h_fc1_drop, W_fc2),ist)
h_fc2 = active_function(BN15)
h_fc2_drop = tf.nn.dropout(h_fc2, keep_prob_MLP)

# Fully connected layer 3 --
# 1024 feature to 18 features.
W_fc_attr= weight_variable([1024, 1000],'attr_layer1')#x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6
b_fc_attr = bias_variable([1000])
fc_attr = tf.matmul(h_fc2_drop, W_fc_attr) + b_fc_attr

W_fc_cate = weight_variable([1024, 50],'cate_layer1')
b_fc_cate = bias_variable([50])
fc_cate = tf.matmul(h_fc2_drop, W_fc_cate) + b_fc_cate

W_fc_landmark = weight_variable([1024, 16],'lan_layer1')#x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6
b_fc_landmark = bias_variable([16])
fc_landmark = tf.matmul(h_fc2_drop, W_fc_landmark) + b_fc_landmark

W_fc_v1 = weight_variable([1024, 2],'vis_layer1')
b_fc_v1 = bias_variable([2])
fc_v1 = tf.matmul(h_fc2_drop, W_fc_v1) + b_fc_v1

W_fc_v2 = weight_variable([1024, 2],'vis_layer2')
b_fc_v2 = bias_variable([2])
fc_v2 = tf.matmul(h_fc2_drop, W_fc_v2) + b_fc_v2

W_fc_v3 = weight_variable([1024, 2],'vis_layer3')
b_fc_v3 = bias_variable([2])
fc_v3 = tf.matmul(h_fc2_drop, W_fc_v3) + b_fc_v3

W_fc_v4 = weight_variable([1024, 2],'vis_layer4')
b_fc_v4 = bias_variable([2])
fc_v4 = tf.matmul(h_fc2_drop, W_fc_v4) + b_fc_v4

W_fc_v5 = weight_variable([1024, 2],'vis_layer5')
b_fc_v5 = bias_variable([2])
fc_v5 = tf.matmul(h_fc2_drop, W_fc_v5) + b_fc_v5

W_fc_v6 = weight_variable([1024, 2],'vis_layer6')
b_fc_v6 = bias_variable([2])
fc_v6 = tf.matmul(h_fc2_drop, W_fc_v6) + b_fc_v6

W_fc_v7 = weight_variable([1024, 2],'vis_layer7')
b_fc_v7 = bias_variable([2])
fc_v7 = tf.matmul(h_fc2_drop, W_fc_v7) + b_fc_v7

W_fc_v8 = weight_variable([1024, 2],'vis_layer8')
b_fc_v8 = bias_variable([2])
fc_v8 = tf.matmul(h_fc2_drop, W_fc_v8) + b_fc_v8
readable_combined=tf.concat(axis=1,
  values=[fc_attr,fc_cate,fc_landmark,fc_v1,fc_v2,fc_v3,fc_v4,fc_v5,fc_v6,fc_v7,fc_v8])
tf.add_to_collection("Readable_MAPs", readable_combined)
return fc_attr,fc_cate,fc_landmark, fc_v1, fc_v2, fc_v3, fc_v4, fc_v5, fc_v6, fc_v7, fc_v8,keep_prob_MLP
```

Figure 5: Major code of Network 2 (part 2)

```
def XceptionNet(x_image,x_supp,ist,cm):

  #===========ENTRY FLOW=============
  #Block 1
  B1_W_conv1 = weight_variable([3,3,3, 32],'B1_W_conv1')
  net=batch_norm(conv2d(x=x_image,W=B1_W_conv1,s=[1,1,2,2],p='VALID'),ist)
  net = active_function(net)
  #print('B1')

  B1_W_conv2 = weight_variable([3,3,32, 64],'B1_W_conv2')
  net=batch_norm(conv2d(x=net,W=B1_W_conv2,p='VALID'),ist)
  net = active_function(net)

  B1_W_conv3 = weight_variable([1,1,64, 128],'B1_W_conv3')
  residual=batch_norm(conv2d(x=net,W=B1_W_conv3,s=[1,1,2,2]),ist)
  #residual = active_function(residual)

  #Block 2
  B2_W1_conv1 = weight_variable([3,3,64, cm],'B2_W1_conv1')
  B2_W2_conv1 = weight_variable([1,1,cm*64, 128],'B2_W2_conv1')
  net=batch_norm(separable_conv2d(x=net,W1=B2_W1_conv1,W2=B2_W2_conv1),ist)
  net = active_function(net)

  B2_W1_conv2 = weight_variable([3,3,128, cm],'B2_W1_conv2')
  B2_W2_conv2 = weight_variable([1,1,cm*128, 128],'B2_W2_conv2')
  net=batch_norm(separable_conv2d(x=net,W1=B2_W1_conv2,W2=B2_W2_conv2),ist)
  net = max_pool_2x2(x=net)
  net=tf.add(net,residual)

  B2_W_conv3 = weight_variable([1,1,128, 256],'B2_W_conv3')
  residual=batch_norm(conv2d(x=net,W=B2_W_conv3,s=[1,1,2,2]),ist)
  #print('B2')

  #Block 3
  net = active_function(net)
  B3_W1_conv1 = weight_variable([3,3,128, cm],'B3_W1_conv1')
  B3_W2_conv1 = weight_variable([1,1,cm*128, 256],'B3_W2_conv1')
  net=batch_norm(separable_conv2d(x=net,W1=B3_W1_conv1,W2=B3_W2_conv1),ist)

  net = active_function(net)
  B3_W1_conv2 = weight_variable([3,3,256, cm],'B3_W1_conv2')
  B3_W2_conv2 = weight_variable([1,1,cm*256, 256],'B3_W2_conv2')
  net=batch_norm(separable_conv2d(x=net,W1=B3_W1_conv2,W2=B3_W2_conv2),ist)

  net = max_pool_2x2(x=net)
  net=tf.add(net,residual)
  B3_W_conv3 = weight_variable([1,1,256,728],'B3_W_conv3')
  residual=batch_norm(conv2d(x=net,W= B3_W_conv3 ,s=[1,1,2,2]),ist)
  #print('B3')
  #Block 4
  net = active_function(net)
  B4_W1_conv1 = weight_variable([3,3,256, cm],'B4_W1_conv1')
  B4_W2_conv1 = weight_variable([1,1,cm*256, 728],'B4_W2_conv1')
  net=batch_norm(separable_conv2d(x=net,W1=B4_W1_conv1,W2=B4_W2_conv1),ist)

  net = active_function(net)
  B4_W1_conv2 = weight_variable([3,3,728, cm],'B4_W1_conv2')
  B4_W2_conv2 = weight_variable([1,1,cm*728, 728],'B4_W2_conv2')
  net=batch_norm(separable_conv2d(x=net,W1=B4_W1_conv2,W2=B4_W2_conv2),ist)

  net = max_pool_2x2(x=net)
  net=tf.add(net,residual)
```

Figure 40: Major code of Network 6 (part 1)

```
#print('B4')
#===========MIDDLE FLOW===============
#1
residual=net
net = active_function(net)
M1_W1_conv1 = weight_variable([3,3,728, cm],'M1_W1_conv1')
M1_W2_conv1 = weight_variable([1,1,cm*728, 728],'M1_W2_conv1')
net=batch_norm(separable_conv2d(x=net,W1=M1_W1_conv1,W2=M1_W2_conv1),ist)

net = active_function(net)
M1_W1_conv2 = weight_variable([3,3,728, cm],'M1_W1_conv2')
M1_W2_conv2 = weight_variable([1,1,cm*728, 728],'M1_W2_conv2')
net=batch_norm(separable_conv2d(x=net,W1=M1_W1_conv2,W2=M1_W2_conv2),ist)

net = active_function(net)
M1_W1_conv3 = weight_variable([3,3,728, cm],'M1_W1_conv3')
M1_W2_conv3 = weight_variable([1,1,cm*728, 728],'M1_W2_conv3')
net=batch_norm(separable_conv2d(x=net,W1=M1_W1_conv3,W2=M1_W2_conv3),ist)
net=tf.add(net,residual)
#print('M1')
#2
residual=net
net = active_function(net)
M2_W1_conv1 = weight_variable([3,3,728, cm],'M2_W1_conv1')
M2_W2_conv1 = weight_variable([1,1,cm*728, 728],'M2_W2_conv1')
net=batch_norm(separable_conv2d(x=net,W1=M2_W1_conv1,W2=M2_W2_conv1),ist)

net = active_function(net)
M2_W1_conv2 = weight_variable([3,3,728, cm],'M2_W1_conv2')
M2_W2_conv2 = weight_variable([1,1,cm*728, 728],'M2_W2_conv2')
net=batch_norm(separable_conv2d(x=net,W1=M2_W1_conv2,W2=M2_W2_conv2),ist)

net = active_function(net)
M2_W1_conv3 = weight_variable([3,3,728, cm],'M2_W1_conv3')
M2_W2_conv3 = weight_variable([1,1,cm*728, 728],'M2_W2_conv3')
net=batch_norm(separable_conv2d(x=net,W1=M2_W1_conv3,W2=M2_W2_conv3),ist)
net=tf.add(net,residual)
#print('M2')
#3
residual=net
net = active_function(net)
M3_W1_conv1 = weight_variable([3,3,728, cm],'M3_W1_conv1')
M3_W2_conv1 = weight_variable([1,1,cm*728, 728],'M3_W2_conv1')
net=batch_norm(separable_conv2d(x=net,W1=M3_W1_conv1,W2=M3_W2_conv1),ist)

net = active_function(net)
M3_W1_conv2 = weight_variable([3,3,728, cm],'M3_W1_conv2')
M3_W2_conv2 = weight_variable([1,1,cm*728, 728],'M3_W2_conv2')
net=batch_norm(separable_conv2d(x=net,W1=M3_W1_conv2,W2=M3_W2_conv2),ist)

net = active_function(net)
M3_W1_conv3 = weight_variable([3,3,728, cm],'M3_W1_conv3')
M3_W2_conv3 = weight_variable([1,1,cm*728, 728],'M3_W2_conv3')
net=batch_norm(separable_conv2d(x=net,W1=M3_W1_conv3,W2=M3_W2_conv3),ist)
net=tf.add(net,residual)
```

Figure 41: Major code of Network 6 (part 2)

```
#print('M3')
#4
residual=net
net = active_function(net)
M4_W1_conv1 = weight_variable([3,3,728, cm],'M4_W1_conv1')
M4_W2_conv1 = weight_variable([1,1,cm*728, 728],'M4_W2_conv1')
net=batch_norm(separable_conv2d(x=net,W1=M4_W1_conv1,W2=M4_W2_conv1),ist)

net = active_function(net)
M4_W1_conv2 = weight_variable([3,3,728, cm],'M4_W1_conv2')
M4_W2_conv2 = weight_variable([1,1,cm*728, 728],'M4_W2_conv2')
net=batch_norm(separable_conv2d(x=net,W1=M4_W1_conv2,W2=M4_W2_conv2),ist)

net = active_function(net)
M4_W1_conv3 = weight_variable([3,3,728, cm],'M4_W1_conv3')
M4_W2_conv3 = weight_variable([1,1,cm*728, 728],'M4_W2_conv3')
net=batch_norm(separable_conv2d(x=net,W1=M4_W1_conv3,W2=M4_W2_conv3),ist)
net=tf.add(net,residual)
#print('M4')
#5



residual=net
net = active_function(net)
M5_W1_conv1 = weight_variable([3,3,728, cm],'M5_W1_conv1')
M5_W2_conv1 = weight_variable([1,1,cm*728, 728],'M5_W2_conv1')
net=batch_norm(separable_conv2d(x=net,W1=M5_W1_conv1,W2=M5_W2_conv1),ist)

net = active_function(net)
M5_W1_conv2 = weight_variable([3,3,728, cm],'M5_W1_conv2')
M5_W2_conv2 = weight_variable([1,1,cm*728, 728],'M5_W2_conv2')
net=batch_norm(separable_conv2d(x=net,W1=M5_W1_conv2,W2=M5_W2_conv2),ist)

net = active_function(net)
M5_W1_conv3 = weight_variable([3,3,728, cm],'M5_W1_conv3')
M5_W2_conv3 = weight_variable([1,1,cm*728, 728],'M5_W2_conv3')
net=batch_norm(separable_conv2d(x=net,W1=M5_W1_conv3,W2=M5_W2_conv3),ist)
net=tf.add(net,residual)
#print('M5')
#6
residual=net
net = active_function(net)
M6_W1_conv1 = weight_variable([3,3,728, cm],'M6_W1_conv1')
M6_W2_conv1 = weight_variable([1,1,cm*728, 728],'M6_W2_conv1')
net=batch_norm(separable_conv2d(x=net,W1=M6_W1_conv1,W2=M6_W2_conv1),ist)

net = active_function(net)
M6_W1_conv2 = weight_variable([3,3,728, cm],'M6_W1_conv2')
M6_W2_conv2 = weight_variable([1,1,cm*728, 728],'M6_W2_conv2')
net=batch_norm(separable_conv2d(x=net,W1=M6_W1_conv2,W2=M6_W2_conv2),ist)
```

Figure 42: Major code of Network 6 (part 3)

```
net = active_function(net)
M6_W1_conv3 = weight_variable([3,3,728, cm],'M6_W1_conv3')
M6_W2_conv3 = weight_variable([1,1,cm*728, 728],'M6_W2_conv3')
net=batch_norm(separable_conv2d(x=net,W1=M6_W1_conv3,W2=M6_W2_conv3),ist)
net=tf.add(net,residual)
#print('M6')
#7
residual=net
net = active_function(net)
M7_W1_conv1 = weight_variable([3,3,728, cm],'M7_W1_conv1')
M7_W2_conv1 = weight_variable([1,1,cm*728, 728],'M7_W2_conv1')
net=batch_norm(separable_conv2d(x=net,W1=M7_W1_conv1,W2=M7_W2_conv1),ist)

net = active_function(net)
M7_W1_conv2 = weight_variable([3,3,728, cm],'M7_W1_conv2')
M7_W2_conv2 = weight_variable([1,1,cm*728, 728],'M7_W2_conv2')
net=batch_norm(separable_conv2d(x=net,W1=M7_W1_conv2,W2=M7_W2_conv2),ist)

net = active_function(net)
M7_W1_conv3 = weight_variable([3,3,728, cm],'M7_W1_conv3')
M7_W2_conv3 = weight_variable([1,1,cm*728, 728],'M7_W2_conv3')
net=batch_norm(separable_conv2d(x=net,W1=M7_W1_conv3,W2=M7_W2_conv3),ist)
net=tf.add(net,residual)
#print('M7')
#8
residual=net
net = active_function(net)
M8_W1_conv1 = weight_variable([3,3,728, cm],'M8_W1_conv1')
M8_W2_conv1 = weight_variable([1,1,cm*728, 728],'M8_W2_conv1')
net=batch_norm(separable_conv2d(x=net,W1=M8_W1_conv1,W2=M8_W2_conv1),ist)

net = active_function(net)
M8_W1_conv2 = weight_variable([3,3,728, cm],'M8_W1_conv2')
M8_W2_conv2 = weight_variable([1,1,cm*728, 728],'M8_W2_conv2')
net=batch_norm(separable_conv2d(x=net,W1=M8_W1_conv2,W2=M8_W2_conv2),ist)

net = active_function(net)
M8_W1_conv3 = weight_variable([3,3,728, cm],'M8_W1_conv3')
M8_W2_conv3 = weight_variable([1,1,cm*728, 728],'M8_W2_conv3')
net=batch_norm(separable_conv2d(x=net,W1=M8_W1_conv3,W2=M8_W2_conv3),ist)
net=tf.add(net,residual)
#print('M8')
#========EXIT FLOW============
E_W_conv1 = weight_variable([1,1,728,1024],'E_W_conv1')
residual=batch_norm(conv2d(x=net,W= E_W_conv1 ,s=[1,1,2,2]),ist)
net = active_function(net)
```

Figure 43: Major code of Network 6 (part 4)

```
E_W1_conv2 = weight_variable([3,3,728, cm],'E_W1_conv2')
E_W2_conv2 = weight_variable([1,1,cm*728, 728],'E_W2_conv2')
net=batch_norm(separable_conv2d(x=net,W1=E_W1_conv2,W2=E_W2_conv2),ist)
net = active_function(net)

E_W1_conv3 = weight_variable([3,3,728, cm],'E_W1_conv3')
E_W2_conv3 = weight_variable([1,1,cm*728, 1024],'E_W2_conv3')
net=batch_norm(separable_conv2d(x=net,W1=E_W1_conv3,W2=E_W2_conv3),ist)
net = max_pool_2x2(x=net)
net=tf.add(net,residual)

E_W1_conv4 = weight_variable([3,3,1024, cm],'E_W1_conv4')
E_W2_conv4 = weight_variable([1,1,cm*1024,1536],'E_W2_conv4')
net=batch_norm(separable_conv2d(x=net,W1=E_W1_conv4,W2=E_W2_conv4),ist)
net = active_function(net)

E_W1_conv5 = weight_variable([3,3,1536, cm],'E_W1_conv5')
E_W2_conv5 = weight_variable([1,1,cm*1536,2048],'E_W2_conv5')
net=batch_norm(separable_conv2d(x=net,W1=E_W1_conv5,W2=E_W2_conv5),ist)
net = active_function(net)
net =tf.nn.avg_pool(net,ksize=[1,1,7,7],strides=[1, 1, 1, 1],padding='VALID',data_format='NCHW')#tf.transpose(,[0, 2, 3, 1])

# #========Conv conection===========
# mark_W_conv1 = weight_variable([1,1,2048,2048],'mark_W_conv1')
# net  = conv2d(x=net,W= mark_W_conv1)#tf.contrib.layers.batch_norm(,is_training=ist,fused=True)
# mark_W_conv2 = weight_variable([1,1,2048,1],'mark_W_conv2')
# logits  = tf.transpose(conv2d(x=net,W= mark_W_conv2), [0, 2, 3, 1])
# logits  = tf.squeeze(logits,axis=[1,2], name='block15_logits')
# final_mark=tf.nn.sigmoid(logits)
# keep_prob_MLP1 = tf.placeholder(tf.float32,name='keep_prob1')#0.5
# keep_prob_MLP2 = tf.placeholder(tf.float32,name='keep_prob2')#0.5


# ================MLP====================
CNN_reshape=tf.reshape(net,[-1,1*1*2048])
combined_layer=tf.concat(axis=1,values=[CNN_reshape,x_supp])
#print(combined_layer.get_shape())

W_MLP1 = weight_variable([1 * 1 * 2048+18, 1024],'MLP_layer1')
net=batch_norm(tf.matmul(combined_layer, W_MLP1),ist)
net = active_function(net)
#print(net.get_shape())

keep_prob_MLP = tf.placeholder(tf.float32,name='keep_prob')#0.5
net = tf.nn.dropout(net, keep_prob_MLP)

W_MLP2  = weight_variable([1024, 1024],'MLP_layer2')
net = batch_norm(tf.matmul(net, W_MLP2),ist)
net = active_function(net)
#print(net.get_shape())

net = tf.nn.dropout(net, keep_prob_MLP)

W_mark = weight_variable([1024, 1],'mark_layer')#x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6
net= batch_norm(tf.matmul(net, W_mark),ist)
final_mark=tf.nn.sigmoid(net)
#print(final_mark.get_shape())
return final_mark,keep_prob_MLP
```

Figure 44: Major code of Network 6 (part 5)