



**The Cat's Pajamas**  
**Fashion Recommendation Systems**

**Final Year Project Report**  
**Name: Hamza Hassan Khan**  
**Student ID: 119401422**  
**Supervisor: Dr Derek Bridge**

**April, 2023**

# Abstract

In our daily lives we interact with the modern world in many domains. The world has evolved in the past few decades resulting in giving us many choices. These choices can be overwhelming for us to make a decision in terms of what to choose as some of the variety might not be relevant to us. For example, a wide range of movies and music genres, variety in modern day fashion. This is where a Recommendation System comes in, it narrows down the choices that a user has to make in a certain domain by making recommendations that takes user's interests and the item's knowledge into account.

In this project we are keen to explore many recommendation techniques in a fashion domain. The dataset comes from a competition that was hosted on Kaggle. It contains H and M (popular clothing brand) transactions, items and product Images. The goal of the project is to experiment by implementing the Recommendation Systems using certain techniques. The Experimentations are evaluated on the basis of their recommendation precision and comparison as to why one can be better than the other. The main techniques that we used in this project are Random items, Most popular Items, TF-IDF (Term Frequency- Inverse Document Frequency), Binary Vectorization, Extracting the embeddings from a Convolutional Neural Network that have been trained images of clothing from the H and M dataset, an Ensemble built by combining through averaging the results from all the Recommenders (except Random and popularity) based on the aforementioned techniques. Evaluation and comparison resulted in the winning Convolutional Neural Network (CNN) based recommendation system having highest precision and thus, being best at making relevant recommendations to the user. In contrast with CNN, the Binary Vectorization based recommendation system came in second place.

# Declaration of Originality

In signing this declaration, you are confirming, in writing, that the submitted work is entirely your own original work, except where clearly attributed otherwise, and that it has not been submitted partly or wholly for any other educational award.

I hereby declare that:

- This is all my own work, unless clearly indicated otherwise, with full and proper accreditation;
- With respect to my own work: none of it has been submitted at any educational institution contributing in any way to an educational award;
- With respect to another's work: all text, diagrams, code, or ideas, whether verbatim, paraphrased or otherwise modified or adapted, have been duly attributed to the source in a scholarly manner, whether from books, papers, lecture notes or any other student's work, whether published or unpublished, electronically or in print.

Signed: .....  .....

Date: ..... 20-04-2023 .....

## Acknowledgements

Special thanks to my supervisor Dr Derek Bridge , for supervising me throughout my final year project, his support and advice played a great role in getting started early all the way to the completion of this project.

# Contents

	<b>5</b>
<b>Chapter 1: Introduction</b>	<b>6</b>
1.1: Recommender Systems	6
<b>Chapter 2 : Analysis of H &amp; M Dataset</b>	<b>8</b>
2.1: Dataset Overview	8
2.2: Product Dataset	8
2.3: Customer Dataset	12
2.4: Transaction Dataset	15
2.5: Sparsity	15
2.6: Image Dataset	15
<b>Chapter 3 : Random and Popularity Recommenders</b>	<b>18</b>
3.1: Design	18
3.2: Implementation	18
3.3: Critique	19
<b>Chapter 4 : Content Based Recommenders</b>	<b>20</b>
4.1: Overview	20
4.2: Binary Vectorization Design	20
4.3: Binary Vectorization Implementation	21
4.4: TF-IDF Design	23
4.5: TF-IDF Implementation	23
4.6: Critique	24
<b>Chapter 5 : Convolutional Neural Network Based Recommendation</b>	<b>25</b>
5.1: Design	25
5.2: Implementation	29
5.3: Critique	37
<b>Chapter 6 : Ensemble Recommendation System</b>	<b>38</b>
6.1: Design	38
6.2: Implementation	38
6.3: Critique	39
<b>Chapter 7 : Evaluation</b>	<b>40</b>
7.1: Experimentation	40
7.2: Random and Popularity Recommendation System	42
7.3: Content Based Recommendation System	42
7.4: Convolutional Neural Network's Activations Based Recommendation System	43
7.5: Ensemble Based Recommendation System	43
7.6: Visual Evaluations.	45

<b>Chapter 8: Conclusion and Future Work</b>	<b>47</b>
8.1: Conclusion	47
8.2: Future Work	47
<b>Bibliography</b>	<b>48</b>

# Chapter 1: Introduction

## 1.1: Recommender Systems

The Recommendation system makes a suggestion to a user in a collection of items. It can also be referred to as a filtering system where relevant items are filtered out of a collection. Relevant means how they are related to the user. These items can be Movies, Music, Fashion etc. A user gives an item a rating to present their interests in it. Which are taken into account when making suggestions. These suggestions can either be personalized or contextualized. Personalized recommenders involve use of the information about the user such as their interests, choices etc about a specific item, this information is stored in the user's profile. Contextualised recommender systems make use of the event based information such as place, time etc.

These systems enable users to see relevant stuff in the entire collection of items or products reducing the information overload. This helps the user in managing their choice. Most of the top Companies such as Netflix, Spotify, Youtube and ecommerce companies etc use recommendation systems to show items that users might like. From a business perspective these systems help to understand what users might be interested in.

Generally how a simple recommender system works is, it takes the user's activity into the account for example Bob watched Spiderman, based on this interaction a movie recommender system might recommend Bob some other marvel movies. Similarly if Alex browsed some high end gaming accessories on an online gaming store, a recommender system might suggest some other parts that are similar to the ones that Alex browsed.

The initial step involves creation of the item profiles in order to get information about the item. Each item in a collection show has its individual profile. For example a movie has features like Its Genre, Rating, Title, Description. The profile should involve the description in words, this can then be compared with another movie to get a similarity score. This other movie can be the one that the user watched. Similarly we create a user profile, which holds the user information that might rely on the user's input on their personal information. However it will be quite difficult for the user to update their activity every single time they interact with the systems, what happens is the system implicitly collects this information as the user interacts with the system[2].

One of the challenges in the recommendation system is the sparsity. This can adversely affect the accuracy of a recommender, Since user and item interaction can be represented using a matrix. In other words sparsity refers to how empty the graph or matrix is. For example in a Movie and User interaction there can be a range of the movies that a user might not have rated that leaves that section of the graph empty. This is one of the major challenges in recommender systems and is an active research area. One of the other problems is, If a fresh user comes along with no profile or previous interaction you might think that the recommender system has nothing to suggest, this is also called a “Cold Start Problem” in this case A recommender system can suggest an item that is liked by most of the users. Scalability is one of the other challenges that is involved; this refers to the ability of the system to cope with the growing information.

Recommendation is an active research area due to that high demand recommender systems. There are ranges of methods that can be used to implement from basic techniques such as random or popularity based recommendations to intermediate techniques such as content based or collaborative filtering to something as complex as use of embedding from a neural network trained on the images along with an ensemble of techniques. Regardless of the advances the recommender systems require to be more accurate, this is what makes Recommendation systems a hot research area. In Recommendation systems domain efforts are made to make these recommenders better and more accurate [1].

Fashion is one of a big business domain, in which retailers are keen to keep up with the modern day fashion trend. This leads to a huge variety in fashion products that can overwhelm a consumer, when it comes to making a choice as some of the items might not be relevant to them. The main idea of the project is iteratively building a series of Fashion Recommender Systems. The Recommenders with the best precision will be the final winning recommenders. This will enable us to experiment how different algorithms are related to each other and how they can be manipulated for better precision. Some of the techniques that we will involve in our experimentation are Term Frequency - Inverse Term Frequency, Binary Vectorization, Activations from Convolutional neural network that classify cloth images and Ensemble. The Dataset is out of competition that took place on kaggle. This is a huge Dataset that is over 34GB. The size, cleaning, analysis and exploration of this data is one of the main challenges of this final year project.



# Chapter 2 : Analysis of H & M Dataset

## 2.1: Dataset Overview

The dataset that is used in this project is from a competition that was held on Kaggle. It is an H&M (Popular Fashion Retailer) dataset. It consists of 9 months of H&M sales transactions, their customer information and their items data with the images. The overall size of the dataset is around 34.56 GB. It contains three Comma Separated Values(csv) files, articles.csv, transaction\_train.csv and customers.csv. The articles (items) and customers files are in thousands of rows, and the transactions file is the largest file with millions of rows. The articles.csv contains all the information about an individual product. The customers.csv file consists of all the data about the customers, finally the transaction\_train.csv file stores all the transaction data, in other words all the purchases of the products that were made by customers. The transactions dataset is the largest after the Images dataset. There are around 32 Million transactions over the period of two years. This contributes to 3.5 GB of one csv file. The transactions date from 2018-09-20 to 2020-09-22. The full dataset also contains images of the items in a directory called images.

Dataset was explored using pandas. The visual charts were created in order to get familiar with the data that we are dealing with. Since the dataset was quite large and would involve a significant amount of hardware resources. Due to the limitations of these resources, One month of data was sampled. This was the last month of the two year period. The analysis below is of the sample dataset since this dataset was used in the entire project.

## 2.2: Product Dataset

Articles file has all the information about the products. There are 26722 Products.

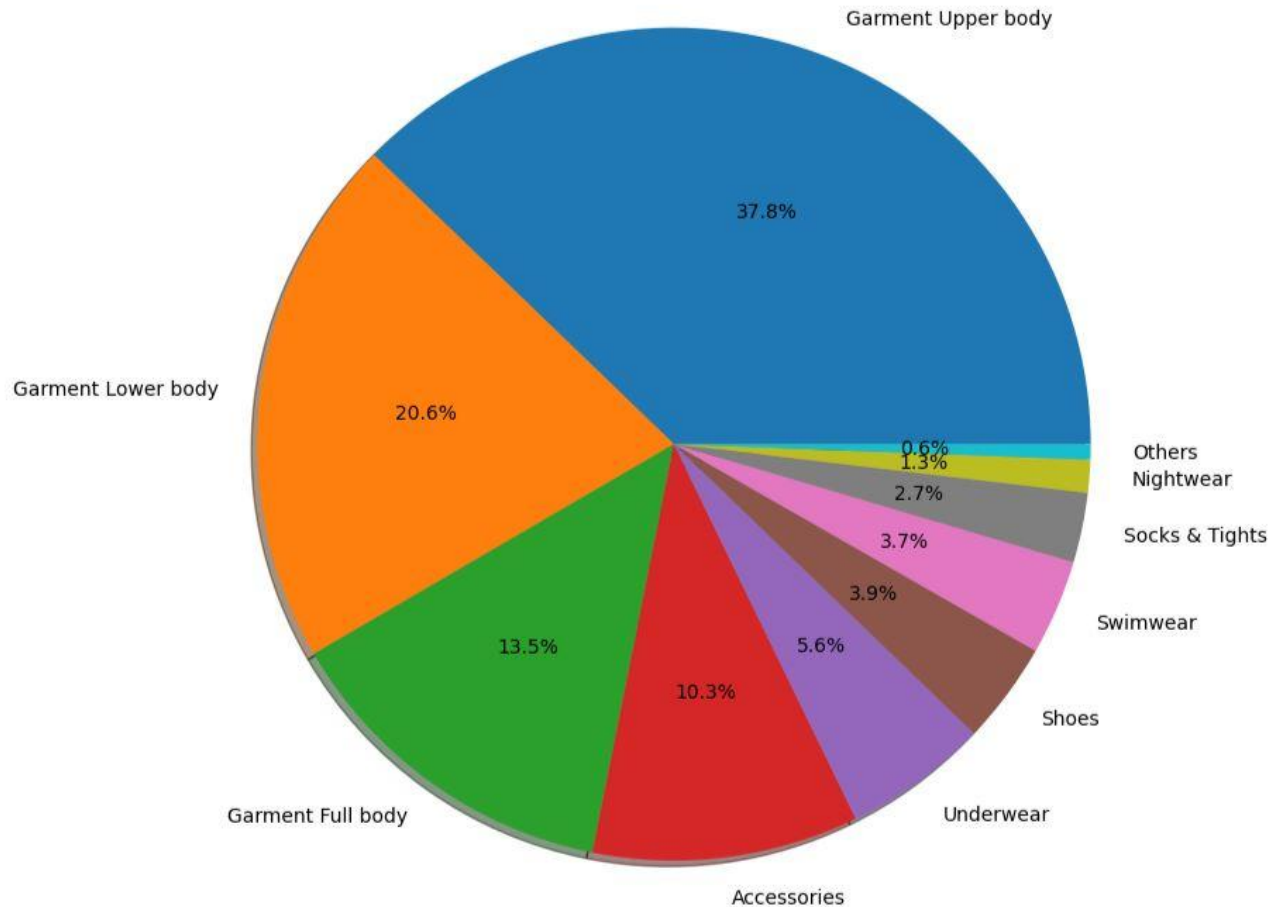
The products have the following attributes.

```
articles_df.columns
```

```
Index(['article_id', 'product_code', 'prod_name', 'product_type_no',  
      'product_type_name', 'product_group_name', 'graphical_appearance_no',  
      'graphical_appearance_name', 'colour_group_code', 'colour_group_name',  
      'perceived_colour_value_id', 'perceived_colour_value_name',  
      'perceived_colour_master_id', 'perceived_colour_master_name',  
      'department_no', 'department_name', 'index_code', 'index_name',  
      'index_group_no', 'index_group_name', 'section_no', 'section_name',  
      'garment_group_no', 'garment_group_name', 'detail_desc'],  
      dtype='object')
```

Those products are categorized in groups.

```
plt.pie(product_group_data, labels = labels, radius=2, autopct='%1.1f%%', shadow=True)
plt.show()
```



Around 40 percent of the products are upper body garments. The pie chart above shows the percentages of the type of cloth. There are 130 classes all together. Some of which are shown in the chart below. These classes refer to the type of the item. The majority class is Trousers.

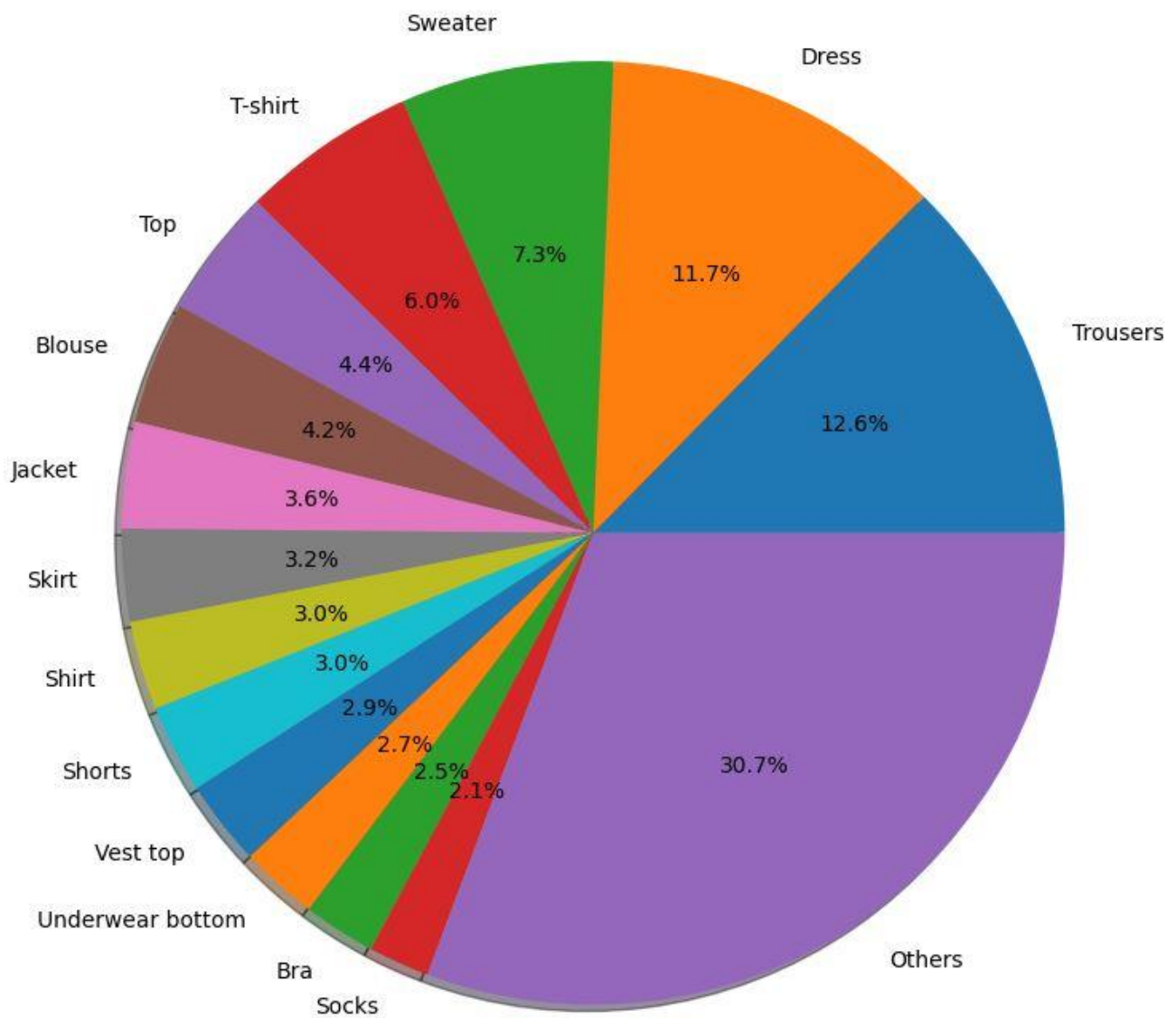
Trousers	3376
Dress	3122
Sweater	1951
T-shirt	1606
Top	1173
Blouse	1120
Jacket	975
Skirt	847
Shirt	815
Shorts	808
Vest top	782
Underwear bottom	715
Bra	663
Socks	559
Others	8210

Name: product\_type\_name, dtype: int64

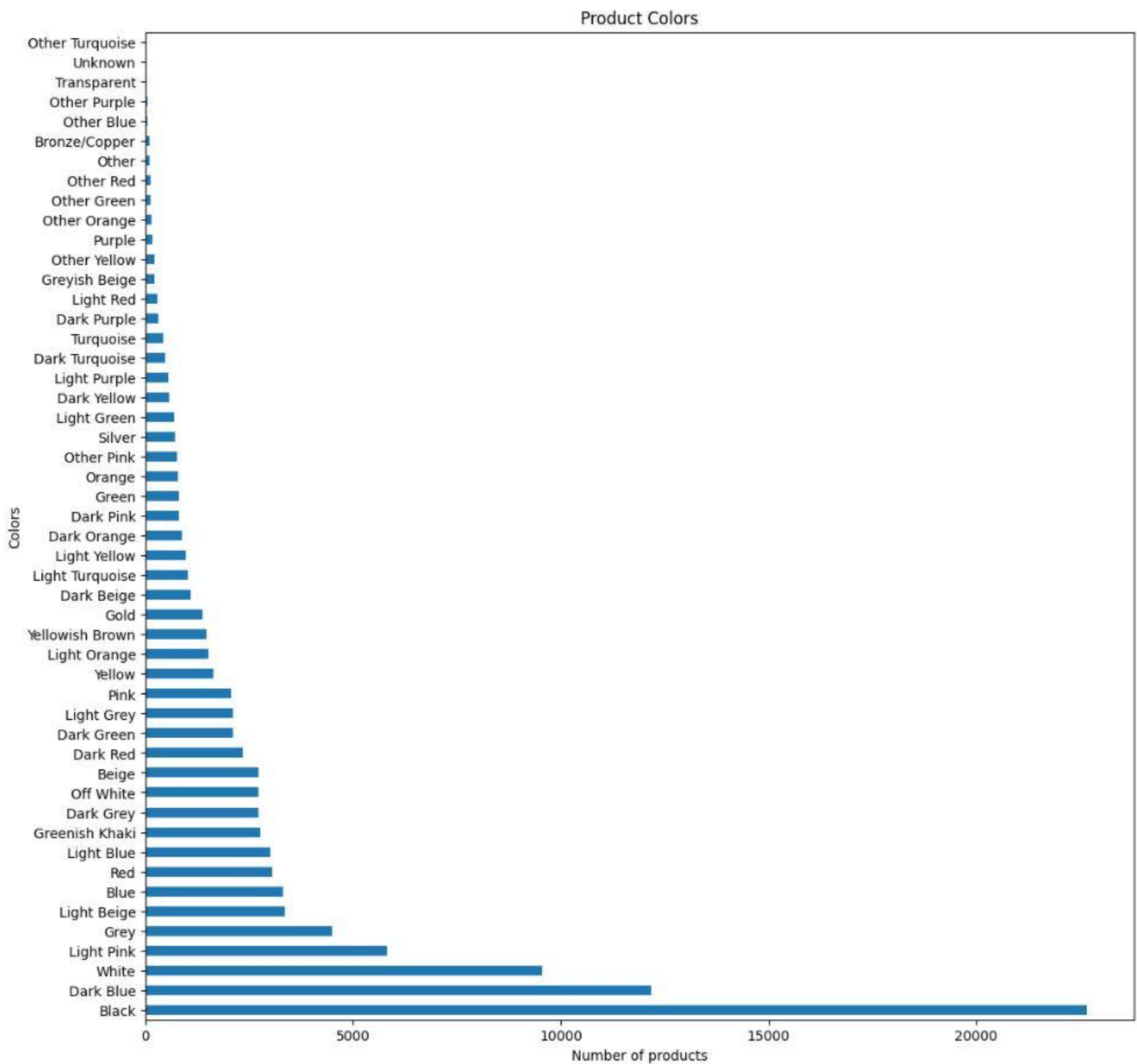
```

colors = list(mcolors.TABLEAU_COLORS.keys())
plt.pie(product_type_data, labels = type_labels, radius=2, autopct='%1.1f%%', shadow=True, colors = colors)
plt.show()

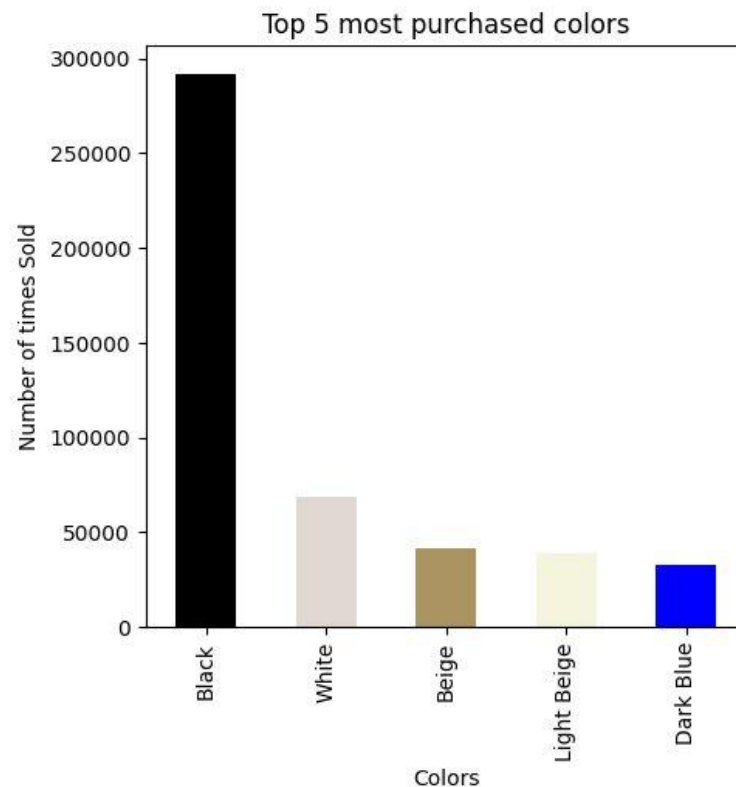
```



The products have around 50 different colors. Most of the products are Black and Dark Blue.



The top 3 most purchased colors are Black, White and Dark Blue



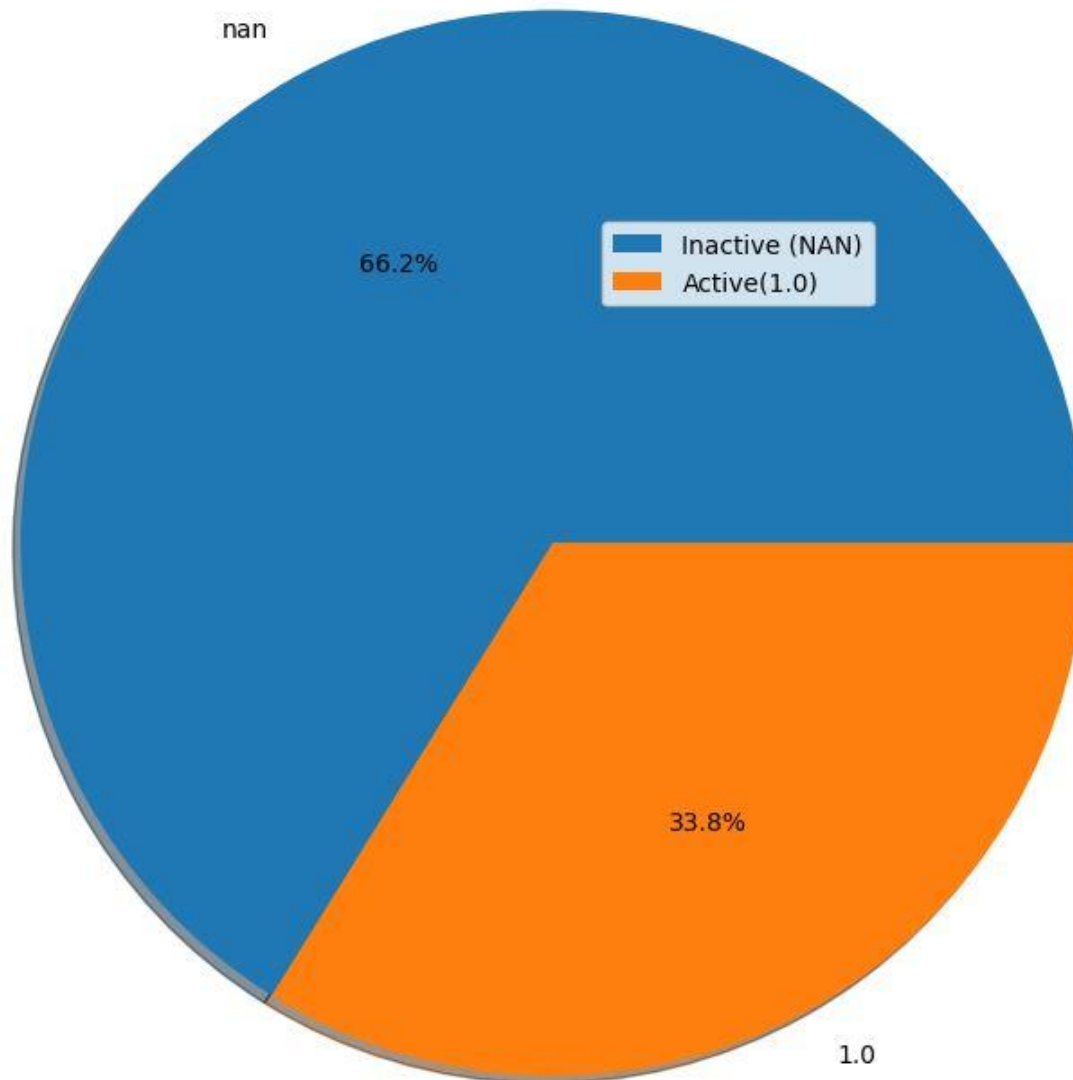
## 2.3: Customer Dataset

The customer's dataset contains information about all the customers and has the following attributes.

```
customers_df.columns  
Index(['customer_id', 'FN', 'Active', 'club_member_status',  
      'fashion_news_frequency', 'age', 'postal_code'],  
      dtype='object')
```

There are around 84234 customers. Each customer has a unique id for identification purposes. This id is included in transactions to identify the customer. There are active and inactive customers. More than half of the customers are inactive. The chart below captures this information.

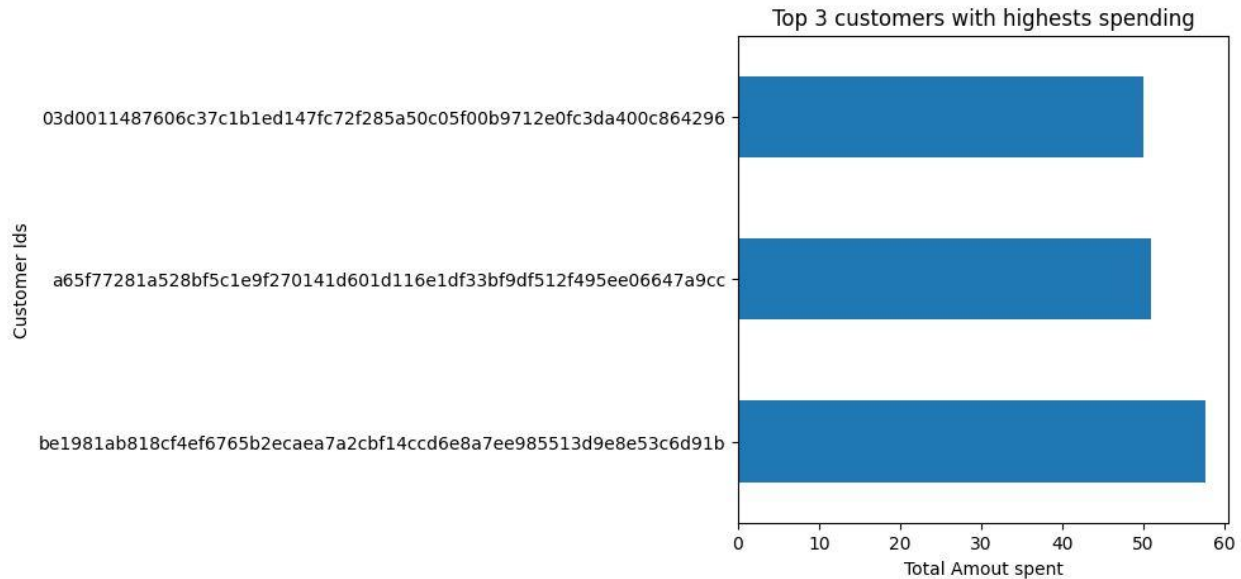
```
active_customers = customers_df['Active'].value_counts(dropna=False)
active_customers_labels = customers_df['Active'].value_counts(dropna=False).keys()
plt.pie(active_customers, labels=active_customers_labels, radius=2, autopct='%1.1f%%', shadow=True)
plt.legend(['Inactive (NAN)', 'Active(1.0)'])
plt.show()
```





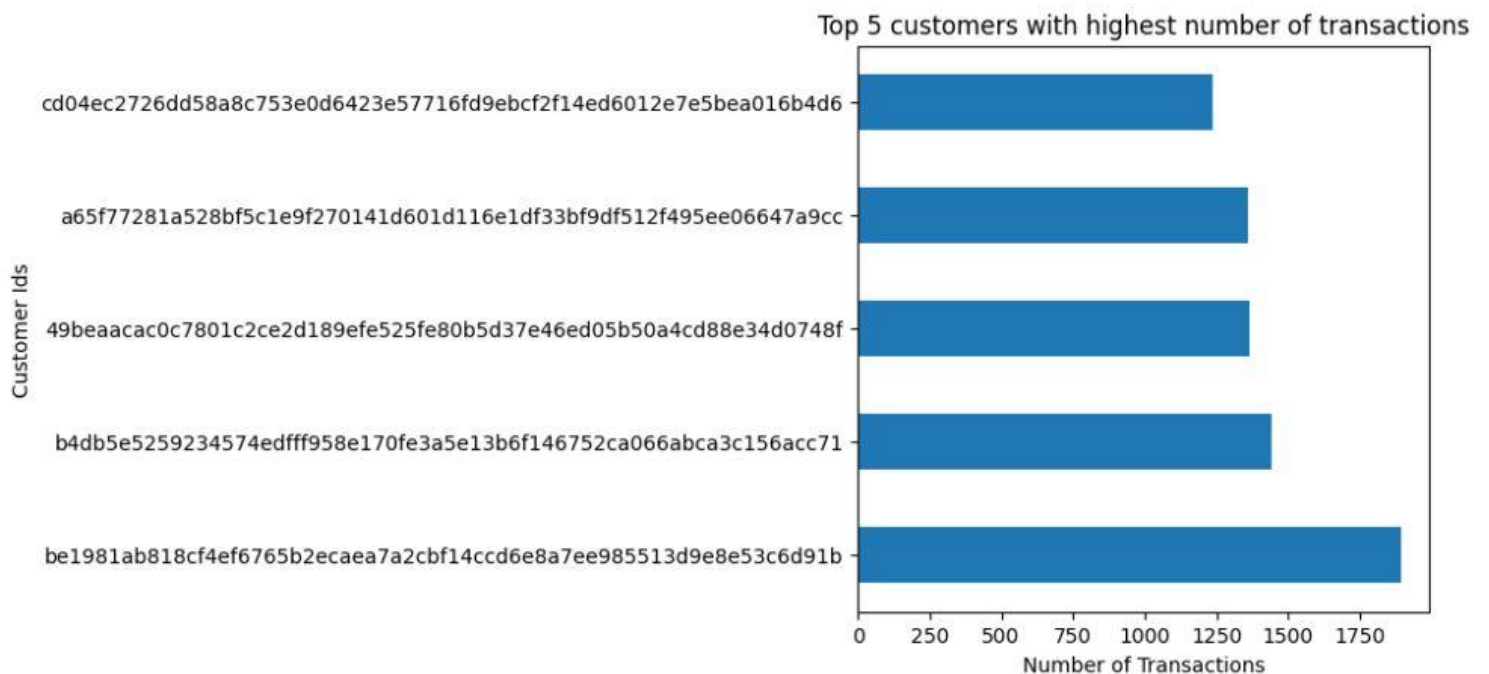
## The customers with highest spending at H and M

```
top_cust = transaction_df.groupby('customer_id')['price'].aggregate('sum').nlargest(3)
top_cust.plot(kind='barh',figsize=(5, 5),title="Top 3 customers with highests spending",ylabel="Customer Ids",xlabel="Total Amou")
plt.show()
# top 3 customers with highests purchases
```



## Customers with Highest number of transactions

```
top_freq_cust = transaction_df['customer_id'].value_counts(dropna=False).nlargest(5)
top_freq_cust.plot(kind='barh',figsize=(5, 5),title="Top 5 customers with highest number of transactions",ylabel="Customer Ids")
plt.show()
```



## 2.4: Transaction Dataset

The transactions are the largest dataset after Images. There are around 788257 transactions over the period of a month. The transactions date from 2020-08-23 to 2020-09-22. The transaction dataset contains important information such as the item ids of all the items that a certain customer has purchased. The customers are identified by unique customer id

```
transaction_df.columns  
Index(['t_dat', 'customer_id', 'article_id', 'price', 'sales_channel_id'], dtype='object')
```

## 2.5: Sparsity

Our customer purchases graph is 96% empty, Sparsity is one of the main challenges in Recommender Systems

```
x = all_products * all_customers
```

Sparsity = 1 - Purchases / Possible Purchases

$$\text{Sparsity} = 1 - \frac{\text{all of the trans actions}}{\text{Number of Customers} * \text{All Products}}$$

```
y = all_transactions/x
```

```
1- y * 100
```

```
0.9649803781592259
```

## 2.6: Image Dataset

The full image dataset contains around 105,000 samples. Each image belongs to a one product and is identified by the id of that product. This dataset was only utilized in the training of Convolutional Neural Network (See Chapter 5). There are around 86 sub-directories that contain images of all the products. The names of the image files correspond to the item's id for identification. The first three digits of the directory correspond to the first three digits of an item id. The picture below shows this structure.

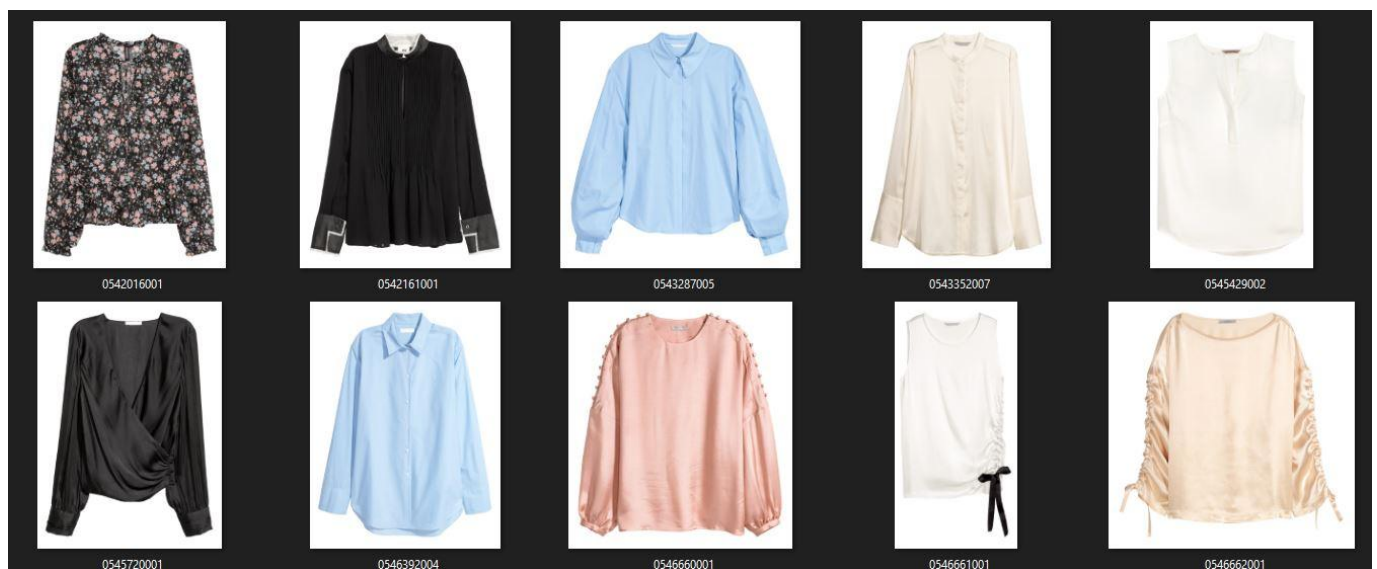
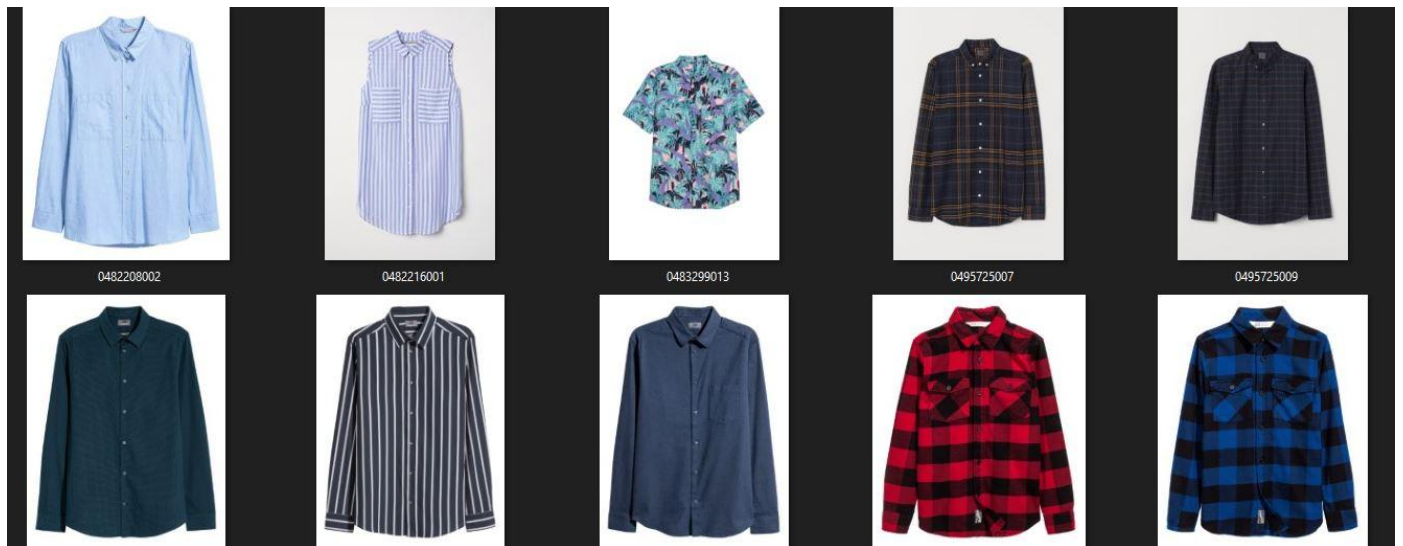


010	624,639	613,180	File folder
011	7,312,768	7,187,039	File folder
012	3,034,946	2,884,970	File folder
013	208,104	205,161	File folder
014	4,671,872	4,528,785	File folder
015	5,368,965	5,289,558	File folder
016	4,664,330	4,503,762	File folder
017	9,019,601	8,761,419	File folder
018	21,802,046	21,220,327	File folder
019	7,910,768	7,772,698	File folder
020	12,410,206	12,213,578	File folder
021	16,926,134	16,517,848	File folder
022	13,275,239	12,973,821	File folder
023	19,789,840	19,389,585	File folder
024	21,337,809	21,018,611	File folder
025	17,546,549	17,007,036	File folder
026	14,614,747	14,220,064	File folder
027	8,790,845	8,492,867	File folder
028	15,892,587	15,430,717	File folder
029	43,855,742	42,889,071	File folder
030	41,226,822	40,358,741	File folder
031	47,662,911	46,724,530	File folder
032	29,615,977	28,919,870	File folder
033	17,119,629	16,618,696	File folder
034	39,623,202	38,611,459	File folder

This structure is not suitable for building a multi-class fashion classifier. The structure is supposed to have directories with a label of a class and all the samples inside belonging to that class. The initial structure above has mixed samples from different classes. For example directory 010 can have trousers, shirts etc. To fix this the entire dataset was restructured to have all the images in their corresponding class directory. There are 130 classes in total. The Image below captures the new structure.

Accessories set	30/01/2023 11:24	File folder
Alice band	30/01/2023 11:23	File folder
Baby Bib	30/01/2023 11:24	File folder
Backpack	30/01/2023 11:24	File folder
Bag	30/01/2023 11:23	File folder
Ballerinas	30/01/2023 11:23	File folder
Beanie	30/01/2023 11:24	File folder
Belt	30/01/2023 11:20	File folder
Bikini top	30/01/2023 11:20	File folder
Blanket	30/01/2023 11:24	File folder
Blazer	30/01/2023 11:23	File folder
Blouse	30/01/2023 11:24	File folder
Bodysuit	30/01/2023 11:19	File folder
Bootie	30/01/2023 11:24	File folder
Boots	30/01/2023 11:20	File folder
Bra	30/01/2023 11:30	File folder

The few images below show the visual representation of the image dataset.



# Chapter 3 : Random and Popularity Recommenders

## 3.1: Design

Random Recommender is the first step towards building a recommendation system. In other words it can be referred to as the “Hello World” of the recommender systems. It simply randomly picks up items and recommends them to the user. It is the worst recommender in terms of precision as it does not take any factors relating to the user that contribute to the recommendation. It is quite trivial to build and gives the starting point. An improved version of recommender is built on top of it, that takes the users preferences or profiles into account when making recommendations.

Popularity Based Recommender on the other hand is the second step and is slightly better than the Random Recommender, it is also one of the solutions to the *Cold Start* problem ( When a fresh user or product has no profile). What happens is instead of randomly recommending products, recommend the products that are most popular among the users. For example, consider a Fashion dataset that has 3 users and 3 items, one of the items is purchased by 2 of users, that item is the most popular one and can be recommended to the third user.

## 3.2: Implementation

The Implementation of both of the recommenders is similar. To build The Random Recommender based on the H and M dataset, randomly pick 5 items from all the products. Below is the code snippet that shows the implementation of a Random Recommender.

```
def random_recommender(customer):  
    # print(f"{customer.customer_id} ", np.random.choice(articles_df['article_id'].values,5))  
    return random.sample(sorted(articles_df['article_id'].values),5)
```

For Implementation of Popularity Recommender, simply pick the top 5 most purchased products. To achieve this consider the transactions dataset, this data set holds all the information about the purchases of the H and M products. By looking at the dataset it is obvious that certain items appear in multiple transactions that were made by multiple users. Using this logic we can use the pandas value\_counts method to group the article\_ids (Product ids) with their count respectively. For example item x appears in 5 transactions then items x was purchased 5 times. Then convert it to a frame and select the ids of the 5 most purchased products. Below is the code snippet of the implementation of a Popularity Recommender

```
def popularity_recommender(customer):  
    # Top 5 most bought  
    popular_products = T['article_id'].value_counts().nlargest(5).to_frame()['article_id'].keys().values  
    return popular_products
```

### 3.3: Critique

Both of those recommendation systems do not take any customer and item interaction into account. Those recommendations might not be relevant to a user at all. Since there is a large transaction and item dataset, an improved recommendation system will take user and item information into account and recommend relevant items to the users to narrow down the variety. This points us to our Content Based Recommendation Systems, that capture the interaction between a customer and the product. This is discussed in detail in our next chapter.

# Chapter 4 : Content Based Recommenders

## 4.1: Overview

Content based recommenders are much improved versions of a recommender system in contrast with popularity and random recommender. Generally content based recommenders take an item profile into the account and score them based on their similarity. The item profile consists of a description of the item in words. This description is then checked for similarity with other items, these items are referred to as candidate items. In this project, Products have a description column that describes a particular item. Candidate Items are the products that the user has not bought. For example if we have a set of 10 products and User has bought 2 then 8 of the products are the candidate items. Calculating the Jaccard which is a similarity measure, is carried by taking the intersection of the set of the items purchased with the candidate items, divided by the union of the purchased and candidate items

We recommend the N items based on the highest similarity scores. The two main design and implementation approaches that we have explored in our project for content based recommenders are **Binary Vectorization** and **TF-IDF** (Term Frequency - Inverse Document Frequency). The precision measure of those two content based recommenders was evaluated however it is not mentioned here for simplicity and is covered in more detail in Chapter 7 (Evaluation).

## 4.2: Binary Vectorization Design

This is a simple content based recommender consider the following example based on our H and M dataset for a better understanding. Each item in the articles.csv is represented by a binary vector of d dimensions. Every element of the vector corresponds to the type of the item, for example trousers. All of those vectors are collected as a matrix Q. Where  $Q^{(i)}$  refers to the column in Q that refers to the item i.

	$i_1$	$i_2$	$i_3$	$i_4$
Trousers	0	1	0	0
Shirt	1	0	0	0
Jacket	0	0	1	0
Shorts	0	0	0	1

The above example is a matrix of a collection of Binary Vectors. For clarification, in the above table the item 1 ( $i_1$ ) is a Shirt and is Not a Trouser, Jacket or a Short. Similarly item 2 ( $i_2$ ) is a Trouser but is not a Shirt jacket or Shorts. This shows a binary value. Where captures the idea of whether or not an item is of a specific type

Consider another table that takes the user's purchases into account. Again this is Matrix P of Binary vectors representing whether or not a user has bought a specific item.  $\mathbf{P}^{(u)}$  corresponds to the row in P that refers to the user U.

	Trousers	Shirt	Jacket	Short
<i>Alice</i>	1	0	0	1
<i>Bob</i>	0	1	1	0
<i>Charlie</i>	0	0	1	0
<i>Darragh</i>	1	1	1	1

In the example above the user's purchases are represented by binary values for example Alice bought Trousers and Shorts. Charlie bought Jacket. Bob Didn't buy the trousers and so on.

Now we have two matrices P and Q, getting the product of these to the matrices will give us a similarity score. We can use many other similarity measures. However in our design we use the cosine similarity measure [2].

### 4.3: Binary Vectorization Implementation

In The Sci-kit learn library we have a **CountVectorizer** class. It takes in many parameters. The two parameters that will be used in the implementation are **stop\_words** and **binary**. The stop\_words will be set to english. So that we only pick up the meaningful words for example not including and, or is etc. Binary parameter is set to True as we want it to act as a binary vectorizer. The method **fit.transform(C)** creates the item profiles. Where C is the column that contains the description of the Products. We then supply the output of the fit\_transform() method to Sci-kit learn's pairwise class's **cosine\_similarity** method. Which returns a similarity matrix with scores representing to what extent one item is similar to another. Below is the visual representation of this matrix.



```
In [9]: count_vectorizer = CountVectorizer(binary=True, stop_words = "english")
profiles= count_vectorizer.fit_transform(i["detail_desc"])
cosine_sim_count = cosine_similarity(profiles)
cosine_count_df = pd.DataFrame(cosine_sim_count, columns=i['article_id'], index=i['article_id'])
cosine_count_df.head(25)
```

```
Out[9]:
```

article_id	775310002	872813003	872813001	860411004	883068004	866755001	399256002	873428004	873428001	883068001	...	893215001	697050011	7838	
article_id	775310002	1.000000	0.264906	0.264906	0.480384	0.348155	0.272166	0.160128	0.372678	0.372678	0.348155	...	0.216506	0.000000	0.0
872813003	0.264906	1.000000	1.000000	0.254514	0.207514	0.594812	0.127257	0.118470	0.118470	0.207514	...	0.229416	0.066227	0.1	
872813001	0.264906	1.000000	1.000000	0.254514	0.207514	0.594812	0.127257	0.118470	0.118470	0.207514	...	0.229416	0.066227	0.1	
860411004	0.480384	0.254514	0.254514	1.000000	0.250873	0.392232	0.230769	0.286446	0.286446	0.250873	...	0.138675	0.000000	0.0	
883068004	0.348155	0.207514	0.207514	0.250873	1.000000	0.142134	0.250873	0.467099	0.467099	1.000000	...	0.000000	0.000000	0.0	
866755001	0.272166	0.594812	0.594812	0.392232	0.142134	1.000000	0.130744	0.121716	0.121716	0.142134	...	0.176777	0.204124	0.1	
399256002	0.160128	0.127257	0.127257	0.230769	0.250873	0.130744	1.000000	0.214834	0.214834	0.250873	...	0.000000	0.000000	0.1	
873428004	0.372678	0.118470	0.118470	0.286446	0.467099	0.121716	0.214834	1.000000	1.000000	0.467099	...	0.000000	0.000000	0.0	

A score of 1 refers to a 1 to 1 similarity between two products, in other words it implies that two items are identical. This can be seen on the diagonal of this matrix. Where the items with the same ids refer to the item itself. However a point to note is that this score of 1 can also be evident in the item that has different ids. For example in the above similarity matrix the third item has a similarity score of 1 with another item with a different id in the second column. This is because those are visually identical items but have a slight visual variation for example two identical items with different colors leading to unique item id but identical item description. This issue of duplicate item description can highly impact the accuracy of this recommendation system, leading to quite optimistic results. This issue is highlighted in detail and is examined through experimentation in detail in the Chapter 7 (Evaluation) the code snippet below shows the implementation of the Binary Vectorization based recommendation system that makes 6 recommendations to a user.

```
: def content_based_recommender_count(customer):
    all_items = pd.DataFrame()
    customer_purchases = t['article_id'][t['customer_id'] == customer].drop_duplicates().values
    for item in customer_purchases:
        sim_i = cosine_count_df[item].sort_values(ascending=False)
        all_items = pd.concat([all_items, sim_i.nlargest(6)])[1:]
    return all_items[0].nlargest(6)
```

## 4.4: TF-IDF Design

TF-IDF stands for term frequency - inverse document frequency. Generally it involves statistical analysis of a particular word appearing in a collection of documents, this presents the importance of a particular word. The TF-IDF is calculated by getting the product of the term frequency and the inverse document frequency.

$$\text{TF-IDF} = \text{Term Frequency} * \text{Inverse Document Frequency}$$

**TF** refers to the frequency of times a word shows up in a certain document divided by the total number of words present in that document. The **IDF** refers to the **log** of the total Documents divided by the total Documents that contain the word. Considering the H and M dataset that is currently being used in this project, the article.csv file that contains all the products, has a column called description (See Chapter 2 for More details). This column contains text that describes the product in words. The TF-IDF technique applied to this column will be used to create the profiles of the items [4].

## 4.5: TF-IDF Implementation

The implementation of the content based recommendations system using TF-IDF is very similar to the previous recommender that was built using the Binary Vectorization idea. The only one subtle difference is that the Sci-Kit Learn Library has a **TfidfVectorizer()** Class [4]. This class will be used in our implementation. We will add the stopwords parameter setting it to English again so that we only pick up meaningful words. After creating the object of the class TfidfVectorizer(), the **fit\_transform(C)** method will be used with the description column given to it [4]. The fit\_transform method will create the candidate profiles by using the TF\*IDF technique as aforementioned above. The rest of the implementation is identical to the Binary Vectorization mentioned above and also carries the duplicated items description issue pointed out above. Below is the snippet of the Implementation and the subtle difference highlighted above.

```
vectorizer = TfidfVectorizer(stop_words = "english")
candidate_profile_X = vectorizer.fit_transform(i["detail_desc"])
```

```
def content_based_recommender_Tfid(customer):
    all_similar_items = pd.DataFrame()
    customer_purchases = t['article_id'][t['customer_id'] == customer].drop_duplicates().values
    for item in customer_purchases:
        similar_items = cosine_df[item].sort_values(ascending=False).nlargest(10)
        all_similar_items = pd.concat([all_similar_items, similar_items.nlargest(6)[1:]])
    return all_similar_items[0].nlargest(6)
```



## 4.6: Critique

Although both of the content based recommendation systems described above are similar in terms of their implementation. The TF-IDF (Term Frequency - Inverse Document Frequency) penalizes the terms that recur across the products. The penalization of terms can be effective in other contexts such as classification where TF-IDF places higher importance on the terms that discriminate. However, in recommendation systems, penalizing a user's tastes for the products (e.g. Trousers) just because many other users like Trousers as well is undesirable as a user's taste for an item is important to be taken into consideration when making other product recommendations. On the other hand, the Binary Vectorization based recommendation system uses binary logic discussed above in its design section, tends to capture more of the customer and the item interaction side, however since the implementation is done using a count vectorizer with binary enabled its downside is its inability to distinguish the importance of one word over the other, It places greater emphasis on abundantly available word[15]. Their evaluations are carried in great detail in Chapter 7, which covers the question, Which one is better in terms of accuracy ?. For the sake of simplicity in understanding implementation and design the evaluations are not mentioned here.

The duplicated descriptions mentioned above can gravely impact the accuracy of those content based recommendation systems. We as humans have an ability to distinguish the subtle variation between two products by visually looking at them even if they are identical in some way. How about if we get our recommendation system to have a similar way of identifying these variations this can make them more accurate than the product description reliant recommenders. Our H and M dataset comes with a huge image dataset, that contains images of the items itself. Visuality of a product would capture more information about it in contrast with a verbal description. We train the recommendation system to be able to classify these images and make recommendations accordingly. This motivates us to examine our next technique that is a Convolutional Neural Network's Activations Based Recommendation System. Its design and implementation and how it differs and is better in comparison with the two techniques described above. This is covered in more detail in the next chapter

# Chapter 5 : Convolutional Neural Network Based Recommendation

## 5.1: Design

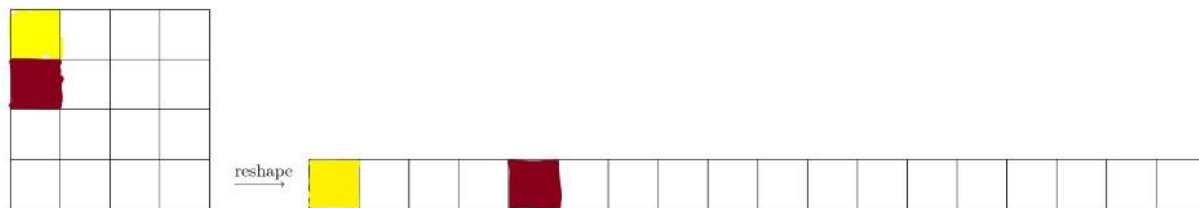
In comparison with the designs experimented in Chapter 4, Convolutional Neural Network based recommendation is one of the most advanced designs approaches. It involves taking the images provided in the H&M dataset into account. Basically, the item image is the input and the embedding is the activation vector that comes out of the second last layer of the a Convolutional Neural Network that was trained on the H&M clothing image dataset. This activation vector is then used to get a cosine similarity score with the other image's activation. This score corresponds to how similar the two images are. The images with highest scores are recommended. The **Initial** Convolutional Neural Network architecture design involved, an input layer followed by one re-scaling layer, Three convolutional layers with 32,64 and 128 feature maps respectively, Two max pooling layers, four dropout layers, one flatten layer, one dense layer and finally an output layer. This resulted in 241,546 trainable parameters. The main reason for the choice of those numbers is to be consistent with James Le's design [5], Below is the visual representation of the architecture.

```
cnn3.summary()
```

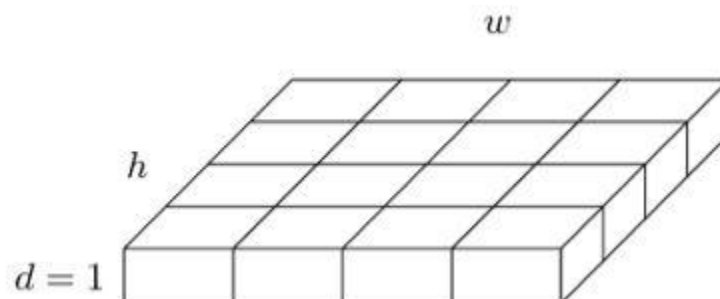
```
Model: "model"
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 28, 28, 1)]	0
rescaling (Rescaling)	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
dropout (Dropout)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
dropout_1 (Dropout)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 128)	73856
dropout_2 (Dropout)	(None, 3, 3, 128)	0
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 128)	147584
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290
Total params: 241,546		
Trainable params: 241,546		
Non-trainable params: 0		

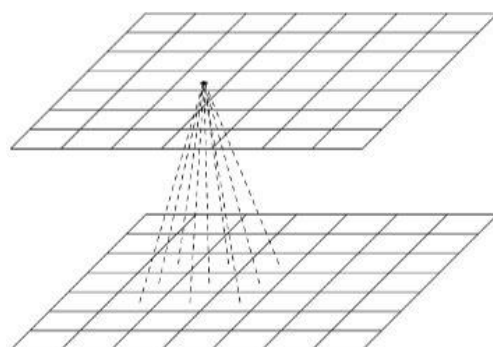
In the summary above it is evident that an input layer is of (28,28,1) shape. This means that an image is 28x28 size and 1 means that it is a grayscale, this image will be taken as input. The details of the images that this network is trained on will be discussed later in the chapter. A two dimensional convolutional layer with shape (h,w,d) is a rank 3 tensor. Since a grayscale image has a height and width, it can be represented as a rank 2 tensor, However when flattening this representation will cause losing the location of a pixel. The figure below shows this disadvantage [6].



The more elegant method is representing the images as a rank three tensor with (h,w,d) to be consistent with color images [6].

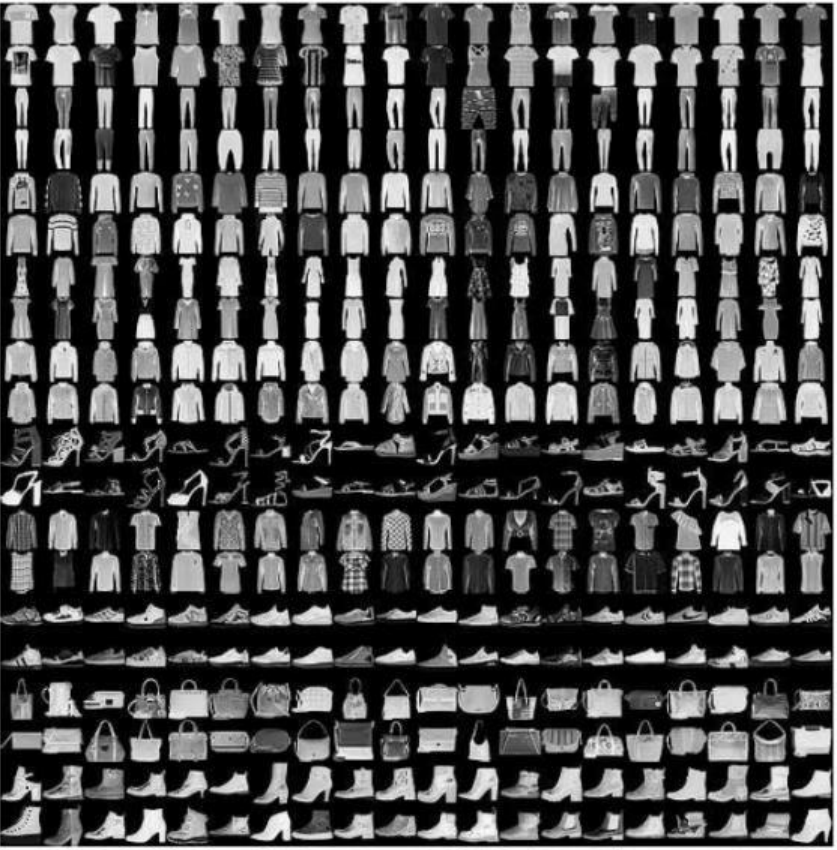


Each layer looks for an area, the output from one layer is used in the next following layer. The upper layers look at the union of what previous layers identified, this enables the upper layer to cover much larger areas. In Convnets the layers are not connected in other words they are not dense in comparison with standard fully connected Neural Networks. Those layers are called Convolutional layers. The neurons in a Convolutional layer connect a small rectangular window of neurons in the preceding layer, a 3x3 window size is used in the design above. The visual representation as follows:



The preceding convolutional layer after the first, loses two pixels to each edge and hence has a shape of  $(h-2, w-2, 1)$ . Convolutional layer also consists of a stack of feature maps. A feature map learns the specific future of its input and it combines several feature maps of previous layers resulting in spatial hierarchy [6].

Pooling layers reduces the number of parameters to be learned resulting in low risk of overfitting. If a pooling layer has a window of a size  $(x, x)$  and the previous layer has a height  $h$  and width  $w$  then the pooling layer will have the shape of  $(h/2, w/2)$ . Pooling layers don't learn anything, in other words they have no weights. A Max pooling layer gives the largest output from the outputs of the previous layer [6]. The dropout layers activate and deactivate the neurons based on the probability set in the dropout rate which reduces the risk of overfitting. Finally there is a dense layer (Fully Connected) 128 neurons followed by another dense layer of 10 neurons; this is the output layer 10 as this Convolutional Neural Network was trained on a Zalando's Fashion MNIST dataset. This dataset has 10 classes and over 60000,  $28 \times 28$  grayscale images. Originally the MNIST dataset contains handwritten digits. Below is the visualization of the Dataset [7].

Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

After Training and Testing the architecture above on Fashion Mnist Dataset. A new design was built, this idea involves the utilization of the training on Fashion Mnist and use it in the H&M dataset of 130 Classes. This is referred to as Transfer Learning where a model can be reused by using it as a base with a few more new layers on the top in other words it can be tweaked and fine tuned according to our problem.

The design was altered by adding one dense layer of 1024 neurons and another dropout layer followed by an outer layer of 130 neurons corresponding to 130 classes. The in-depth analysis of the H&M image part of the dataset is covered in detail in Chapter 2. Below is the new model.

Model: "model\_1"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 28, 28, 1)]	0
rescaling_1 (Rescaling)	(None, 28, 28, 1)	0
model (Functional)	(None, 10)	241546
flatten_1 (Flatten)	(None, 10)	0
dense_2 (Dense)	(None, 1024)	11264
dropout_4 (Dropout)	(None, 1024)	0
dense_3 (Dense)	(None, 130)	133250
=====		
Total params: 386,060		
Trainable params: 144,514		
Non-trainable params: 241,546		

The activation vector comes from the dense\_2 layer with 1024 activations. This vector is utilized for a cosine similarity scale and making recommendations, based on the score. The new model's base is frozen so it won't lose what it has learned, this is why the trainable parameters of our new model are 144,514.

In Comparison with the design of Content Based Recommendation Systems mentioned in Chapter 4, The Convolutional Neural Network's image activation based recommendation is much more sophisticated as it involves a series of steps such as

getting the activations of thousands of images, significant training, validation and testing. For example we trained it on Fashion Mnist first and then created a new model tweaked by using the trained model as base to use it on our main H&M image dataset. There is a lot of computation involved and also considerable knowledge of Neural Network in general to implement a Convolutional Neural Network architecture and on top of that, the huge amount of time involved in the full implementation of the design and fetching the image activations out of the dense layer. On the other hand, the TF-IDF (Term Frequency and Inverse term frequency ) approach involves a very simplistic design. It takes the description of the products and creates a tf-idf vector, which is then used to get similarity scores. Similarly the Binary Vectorization recommender system design uses a count vectorizer to create the vector using the item descriptions and then it is simply a product of two matrices to get the scoring. Those content based recommendation systems are significantly low time consuming in computation and are easier to implement than the Convolutional Neural Network based recommender system.

## 5.2: Implementation

In contrast with the implementation of the Term Frequency and Document Inverse Frequency (TF-IDF) and Binary Vectorization based recommendation systems described in Chapter 4, The convolutional neural network based recommendations system is quite sophisticated to implement as it involves building a convolutional neural network from scratch and training. Initially the Fashion Mnist Dataset was imported and loaded from Keras datasets. This dataset comes with a training and a test set, with 60,000 examples for training and 10000 for test, additionally this dataset was reshaped to represent it as a rank 3 tensor with (height,width,depth) in order to be consistent with our aforementioned design approach. Consider the code snippet below that corresponds to our Convolutional Neural Network Model.

```
inputs = Input(shape = (28,28,1))
x = Rescaling(scale=1./255)(inputs)
x = Conv2D(filters=32, kernel_size=(3, 3), activation='relu')(x)
x = MaxPooling2D((2, 2))(x)
x = Dropout(0.25)(x)

x = Conv2D(filters=64, kernel_size=(3, 3), activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Dropout(0.25)(x)

x = Conv2D(filters=128, kernel_size=(3, 3), activation='relu')(x)
x = Dropout(0.4)(x)

x = Flatten()(x)

x = Dense(128, activation='relu')(x)
x = Dropout(0.3)(x)
outputs = Dense(10, activation='softmax')(x)
cnn3 = Model(inputs,outputs)
```



The keras library's layer class provides the methods to create the Input, Convolutional, Max Pooling, dropout, Rescaling, Flatten and Dense layers in our model. In our three convolutional layers, the filter corresponds to the number of feature maps chosen by us. In our implementation approach we choose 32,64 and 128 numbers of feature maps respectively. The model class then allows us to create our model by grouping these layers together creating a Convolutional Neural Network. The kernel\_size refers to the small rectangular window mentioned above in our design, to which the neurons in the convolutional layer are connected to, and is set as 3x3 in our model. Rectified Linear Unit (ReLU) is used as activation function for all the convolutional and dense layers other than the output layer. ReLU helps with the vanishing gradient problem and results in faster models. ReLU is popularly used as a default activation function, it simply outputs a 0 if the supplied value is negative and it outputs the value itself if it is greater than or equal to zero[8].

### Rectified Linear Unit : $f(x) = \max(0, x)$

The output layer is a dense layer with 10 neurons. As the name suggests these are the outputs, since there are 10 classes in the Fashion MNIST dataset the output from the 10 neurons corresponds to the probability of how likely the input image can be of a certain class. Unlike the other layers the output layer has the softmax activation function. This is because we are doing a multiclass classification (10 Classes ). Softmax activation function normalizes the weighted sum to probabilities that add up to one [9].

### Softmax

$$Prob(\hat{y} = c | \mathbf{x}) = \sigma(\mathbf{x}\beta_c) = \frac{e^{\mathbf{x}\beta_c}}{\sum_{c' \in C} e^{\mathbf{x}\beta_{c'}}}$$

Finally our Convolutional Neural Network Model was compiled at a low learning rate. Choosing the learning rate can be a bit tricky. While training, too large learning rate can result in sub optimal values on the other hand setting the learning rate with too small value can lead to more time consumption. In our model compilation we set it small so that our model learns better during the training. Sparse Categorical Cross Entropy is our loss function as we are doing a multi-class classification. Sparse because the classes are integers. Consider the figure below taken from Tensorflow's Fashion Mnist documentation capturing this detail [10].

Feature	Class	Shape	Dtype	Description
FeaturesDict				
image	Image	(28, 28, 1)	uint8	
label	ClassLabel		int64	

The snippet below corresponds to compilation of our Convolutional Neural Network with the configuration mentioned above [11].

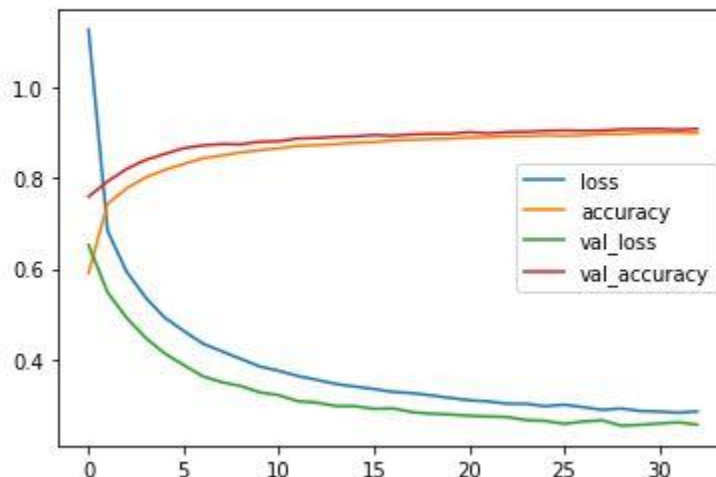
```
cnn3.compile(optimizer=RMSprop(learning_rate=0.0001), loss="sparse_categorical_crossentropy", metrics=["accuracy"])
```

At this point our model is ready for training and testing. The Fashion Mnist dataset came with only the training and the test set. The test set is never looked at and its main goal is to see how well the model performs on data it has not seen. However Kears' .fit method takes a parameter called validation\_split , this enables us to use a proportion of the dataset as a validation set. This was set to 0.2, implying a 20% validation portion. The other parameters that were added to the fit method were the number of epochs, and Early Stopping callback. One Epoch is when entire training data is used i.e all the samples in one cycle. Early stopping is used to reduce the time consumption during training. It stops the training early at a certain epoch, if the loss stops improving, There were three parameters used in the Early Stopping method monitor set to values loss, this implies that we are monitoring the loss value, patience was set to 4, this is the number of epochs we wait before stopping when the error does not improve, finally restore best weights is set to true to keep the best weights. The code snippet below captures this configuration [11].

```
start = time.time()
cnn3_history = cnn3.fit(x_train, y_train, epochs=50,
                       validation_split=0.2,
                       callbacks=[EarlyStopping(monitor="val_loss", patience=4, restore_best_weights=True,
                                                verbose=1)
                                ],
                       verbose=1)
print("Total time: ", time.time() - start, "seconds")
```

An accuracy of 91% was achieved on the training dataset and 90% on the test set using the configuration above. Consider the visual representation below.

```
In [ ]: pd.DataFrame(cnn3_history.history).plot()
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa89030e670>
```





```
In [ ]: train_loss, train_acc = cnn3.evaluate(x_train,y_train)
        train_acc

1875/1875 [=====] - 7s 4ms/step - loss: 0.2281 - accuracy: 0.9174
Out[15]: 0.917400024795532

In [ ]: test_loss, test_acc = cnn3.evaluate(x_test,y_test)
        test_acc

313/313 [=====] - 1s 4ms/step - loss: 0.2645 - accuracy: 0.9057
Out[16]: 0.9057000279426575
```

This model implementation was based on a model architecture taken from James Lee's jupyter. Initially an old method of implementation used in his approach, the model was tweaked using the configuration above [5].

Now a Fashion Mnist trained and tested Convolutional Neural network is ready. At this stage using this model as a base we add a few more layers on the top and implement a new Convolutional Neural Network classifying the fashion images of 130 classes. The H&M dataset was partitioned into Training, Test and Validation sets. Those sets are stored in their corresponding directories and in each directory 130 sub-directories correspond to the 130 classes. Furthermore the actual images were read in as 28x28 gray scale images, to be consistent with the Fashion Mnist trained base. This was done by reading each of the training, validation and test directories. Keras preprocessing class's `image_dataset_from_directory` enables us to achieve this [11].

```
= image_dataset_from_directory(directory=train_dir, label_mode="categorical",color_mode='grayscale', image_size=(28,28))
image_dataset_from_directory(directory=val_dir, label_mode="categorical",color_mode='grayscale', image_size=(28, 28))
image_dataset_from_directory(directory=test_dir, label_mode="categorical",color_mode='grayscale', image_size=(28, 28))

Found 3045 files belonging to 130 classes.
Found 1499 files belonging to 130 classes.
Found 1575 files belonging to 130 classes.
```

Due to hardware resources limitations Google Colab was used in training, validation and testing of this Fashion Image Classifier. The dataset was sampled from each class and then it was uploaded on the google drive. Overall 6119 images samples were taken which was then splitted into training test and validation resulting in the numbers in the picture above.

Figure below shows the new implementation.

```

inputs = Input(shape = (28,28,1))
x = Rescaling(scale=1./255)(inputs)
x = cnn3(x)
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.3)(x)
outputs = Dense(130, activation='softmax')(x)
cnn3_new = Model(inputs=inputs,outputs=outputs)

```

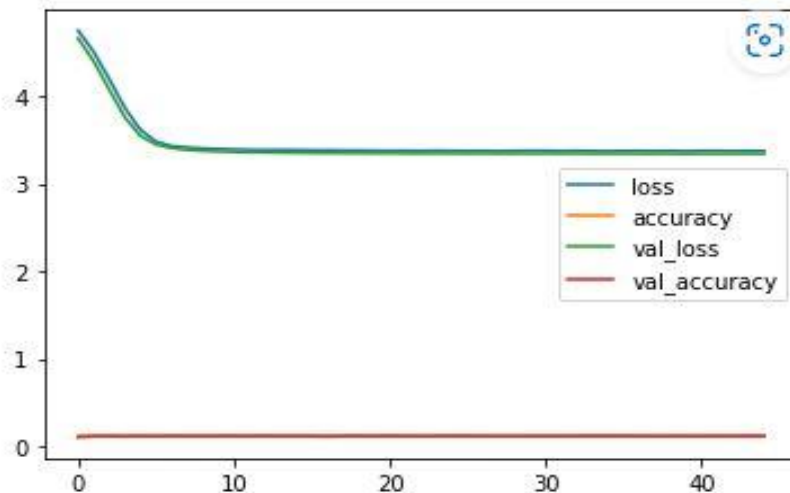
The weights inside the layers of the convolutional base are frozen. So those features that our model learned already are not lost [11].

*cnn3.trainable = False*

The new model was compiled with one difference in contrast with the old Fashion Mnist model. This difference was instead of using the sparse categorical entropy as a loss function, categorical cross entropy was used as our classes are no longer labeled as integers (sparse). Now each individual class has a label and we have 130 in total these are one-hot encoded.

Finally we use fit with the same configuration. The model did not do very well as the layers of the convolutional base were frozen. The visual representation is provided below

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa833318460>

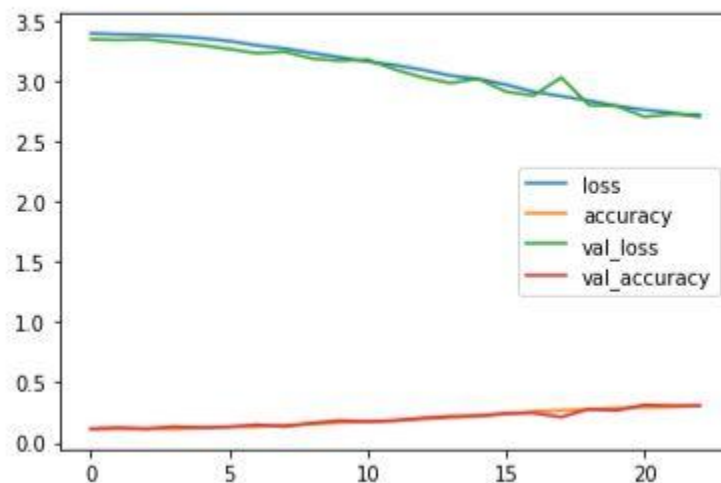


After the training the weights were unfrozen

*cnn3.trainable = True*

The learning rate was re-adjusted to 0.001 instead of 0.0001 to make the training a bit faster. The model was recompiled and trained. The graph below shows the results of training and validation of our Fashion Classifier.

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa8338f2d60>



Training and validation loss can be seen to be coming down and accuracy going up.

Finally, the Fashion Classifier was evaluated on training test and validation sets, below are the results.

```
train_loss, train_acc = cnn3_new.evaluate(train_dataset)
train_acc
```

```
96/96 [=====] - 24s 218ms/step - loss: 2.6938 - accuracy: 0.3067
0.3067323565483093
```

```
val_loss, val_acc = cnn3_new.evaluate(val_dataset)
val_acc
```

```
47/47 [=====] - 12s 187ms/step - loss: 2.7077 - accuracy: 0.3135
0.31354236602783203
```

```
test_loss, test_acc = cnn3_new.evaluate(test_dataset)
test_acc
```

```
50/50 [=====] - 384s 6s/step - loss: 2.8634 - accuracy: 0.2857
0.2857142984867096
```

High accuracy was not expected because the number of classes is 130, but how do we know if the Fashion Classifier that we built is any good ?. This is done by comparing our Fashion Classifier with the Majority Class Classifier that just outputs the majority class. According to our data analysis the majority class is trousers (See Chapter 2). In our sample data set we had 706 trousers samples including all in the training, validation and test sets. So dividing the majority class over the total number samples will give the majority classifier's evaluation. Consider the code snippet below.

```
Total_products = 3045 + 1499 + 1575
Majority_Class = 174 + 174 + 358 # Trousers
Target = Majority_Class/Total_products * 100 # entire dataset
Target
# we are doing better than the Majority Class
```

11.537832979244975

The Majority Class Classifier tends to have an accuracy of 11.5% in comparison to our Fashion Classifier that has an accuracy of 28%. This implies that we are doing better. Due to hardware resources limitations, time consumption. This fashion classifier was accepted to be the best we can do in the light of the circumstances.

At this stage we are now ready to build a fashion recommendation system that uses the activations from a Convolutional Neural Network to make recommendations. Consider the new model's architecture image provided in the design section above. Initially the activation vectors for each image were taken from the dense\_2 layer above the dropout layer with 1024 neurons, this means we will have 1024 activations in our embedding. These activation vectors were taken for every image in the dataset, around 26000 overall. These activation vectors were stored in pandas dataframe which was then stored as embeddings csv. This csv file was used then to create a cosine similarity matrix. Every row is of 1024 entries followed by an image id column that refers to the image to which the embedding belongs. The figure below captures the visual representation.

	0	1	2	3	4	5	6	7	8		1015	1016	1017	1018	1019	1020	1021	1022	1023	image
0	0.049087	0.0	0.030510	0.000000	5.933424e-02	0.000000	0.0	0.025057	0.138850		0.003049	0.011435	0.0	0.000000	0.126702	0.134425	0.000000	0.069312	0.111229	775310002
1	0.000000	0.0	0.037686	0.000000	6.781805e-02	0.064060	0.0	0.000000	0.000000		0.086035	0.000000	0.0	0.148402	0.067697	0.042614	0.000000	0.000000	0.010147	872813003
2	0.000000	0.0	0.031263	0.000000	7.051123e-02	0.065301	0.0	0.000000	0.000000		0.083533	0.000000	0.0	0.133070	0.069236	0.046167	0.000000	0.000000	0.015295	872813001
3	0.021971	0.0	0.032965	0.000000	4.485123e-02	0.035633	0.0	0.000000	0.075174		0.022164	0.000000	0.0	0.000546	0.103162	0.059424	0.000000	0.019324	0.069340	860411004
4	0.000000	0.0	0.066148	0.000000	4.568420e-02	0.000000	0.0	0.041443	0.122582		0.046436	0.000000	0.0	0.000000	0.102105	0.114178	0.000000	0.000000	0.101119	883068004
5	0.005006	0.0	0.030079	0.000000	6.540348e-02	0.070500	0.0	0.000000	0.002131		0.077954	0.000000	0.0	0.119717	0.069923	0.039694	0.000000	0.000000	0.020130	866755001
6	0.000000	0.0	0.178052	0.000000	0.000000e+00	0.194402	0.0	0.000000	0.094213		0.071223	0.000000	0.0	0.199016	0.000000	0.000000	0.000000	0.000000	0.012902	399256002
7	0.000000	0.0	0.073244	0.000000	9.485602e-03	0.032237	0.0	0.009363	0.123628		0.033791	0.000000	0.0	0.000000	0.088291	0.058222	0.000000	0.000000	0.087588	873428004
8	0.000000	0.0	0.079281	0.000000	1.798546e-02	0.019293	0.0	0.024903	0.125762		0.045649	0.000000	0.0	0.000000	0.088570	0.076288	0.000000	0.000000	0.093231	873428001
9	0.000000	0.0	0.062098	0.000000	5.504129e-02	0.000000	0.0	0.051582	0.141500		0.041916	0.001014	0.0	0.000000	0.110011	0.141136	0.000000	0.017717	0.116051	883068001



The cosine similarity matrix was created by using the image id column for indexing and then dropping it. This matrix captures the idea of how similar one image is to another. If it is an identical image then we will have a similarity score of 1.0. This is evident diagonally. The image below shows the visual representation of this matrix.

```
] cosine = cosine_similarity(embeddings_df.drop('image',axis=1))
images = embeddings_df['image'].values
cosine_df = pd.DataFrame(cosine, columns=images, index=images)
cosine_df.head(25)
```

	775310002	872813003	872813001	860411004	883068004	866755001	399256002	873428004	873428001	883068001	...	806261002	893215001
775310002	1.000000	0.561984	0.454619	0.492921	0.491716	0.236495	0.152234	0.261877	0.245796	0.272522	...	0.000058	0.000072
872813003	0.561984	1.000000	0.944661	0.790820	0.722532	0.741748	0.690243	0.639194	0.623160	0.603402	...	0.457105	0.457113
872813001	0.454619	0.944661	1.000000	0.929606	0.878210	0.919415	0.853634	0.847321	0.836406	0.821237	...	0.718954	0.718959
860411004	0.492921	0.790820	0.929606	1.000000	0.978373	0.954371	0.883435	0.953942	0.945616	0.941945	...	0.839326	0.839329
883068004	0.491716	0.722532	0.878210	0.978373	1.000000	0.935744	0.873853	0.965164	0.962396	0.966278	...	0.858353	0.858361
866755001	0.236495	0.741748	0.919415	0.954371	0.935744	1.000000	0.942030	0.976309	0.973725	0.963978	...	0.933764	0.933765
399256002	0.152234	0.690243	0.853634	0.883435	0.873853	0.942030	1.000000	0.933833	0.928990	0.904753	...	0.889589	0.889578

Finally the convolutional neural network's activations based recommender was implemented. The matrix above was queried to get similar items to what a user has bought. Top 6 most similar items to each individual item that a user has bought were stored in a pandas dataframe, not including the exact purchased item. After all the user purchases were taken into account. Items with top 5 highest scores were recommended to the user. The code snippet below shows this implementation. The items that had no image were simply ignored [12].

```
def cnn_recommender(customer):
    default_vals = cnn_cosine_df[775310002].sort_values(ascending=False).head(1)
    default_vals.iloc[0] = 0.0
    all_i_df = pd.DataFrame()
    all_i_df = pd.concat([all_i_df,default_vals])
    customer_purchases = t['article_id'][t['customer_id'] == customer].drop_duplicates().values
    for item in customer_purchases:
        cls = i['product_type_name'][i['article_id'] == item].values[0]
        if os.path.exists(f'../datasets/images/{cls.replace("/","")}/{0{item}.jpg}'):
            similar_imgs = cnn_cosine_df[item].sort_values(ascending=False)
            all_i_df = pd.concat([all_i_df,similar_imgs.nlargest(6)])[1:]
    return all_i_df[0].nlargest(6)
```

To simply test if our recommendation system is functional we passed in a random customer to see if our system makes recommendations for that user. The question of whether or not the customer actually purchased any of those recommended items?, is covered in more detail in the evaluation chapter. Those recommendations were not evaluated and a few are given below visually.

```
print(cnn_recommender('65cb62c794232651e2ac711faa11c2b4e3d41d5f3b59b50bee3ffde1d5776644'))
```

[859076001, 881544005, 859081002, 854222002, 839402002]



### 5.3: Critique

In contrast with the TF-IDF(Term Frequency- Inverse Document Frequency) and Binary Vectorization based recommendation systems. The Convolutional Neural Network Activations based recommendation system is quite difficult to implement as it involves a significant amount of training, computation and time. The embeddings matrix is used in creating a cosine similarity matrix, this makes it fast to fetch the similar images as we can use the index of a purchased item and find the items that are similar to it.

However since fashion is a large business domain, the retailers are enthusiastic to keep up to date with the modern day fashion trend and hence the domain is evolving rapidly with new products coming in frequently. This means every time a pile of new products comes in their images are to be passed in the convolutional neural network to get the activations and then concatenate them to existing csv file with all the embeddings and finally update the cosine similarity matrix. This approach is still better as the wait time for users is significantly reduced as we are querying the cosine similarity matrix rather than making the user wait and find the similarities by getting activation from the individual image of each of the user purchases in the backend.

On the other hand our content based recommender has a much simplistic implementation. The TF-IDF or Binary vectors can be updated by calling the `fit_transform` method on the detail description column that involves new item descriptions. The cosine similarity matrix can be created and the implementation does not involve any training, testing or making an embedding matrix while going through every image, furthermore the content based recommendations systems does not involve the need of using a GPU. The Convolutional Neural Network's activations based recommendation require a large hardware infrastructure that comes with a high cost. For example in the implementation above the data was sampled then uploaded to google drive so it can be trained, validated and tested in google colab using a GPU. But the question stands if this Recommendation System is worth the effort, this is covered in more detail in the evaluation chapter. But Can we do better? One answer could be: why not combine the techniques that we have explored previously and build an Ensemble. This motivation is covered in the next chapter.

# Chapter 6 : Ensemble Recommendation System

## 6.1: Design

There are a range of designs that we have explored in the previous chapters. Why not combine them to create a Recommendation System. The combination of algorithms is known as Ensembles. The idea is to combine the recommendations from our Content based recommendation system and our Convolutional Neural Network Activations based recommendation system and average the score of an individual recommendation. This design approach is simplistic as we already have other recommendation systems implemented. The other Ensemble based recommendation design is simple one step ahead of this initial approach. This other design approach involves allocating weights to the scores of the recommendation from all other recommendations systems [13].

## 6.2: Implementation

The TF-IDF (Term Frequency -Inverse Document Frequency) and Binary vectorization based recommendation systems along with the Convolutional Neural Network Activations based recommendation system, were tweaked to take in an item id as input rather than the customer id. Each of our recommendation systems outputs an item similar to the input item. The scores of all the recommendations made by the combined recommendation systems were averaged and stored in a pandas dataframe that was done for each item that user has bought. Finally the top 5 items with highest scores were recommended to the user. In a weighted ensemble each recommendation is multiplied by a selected weight that is covered in more detail in the Evaluation chapter. Below is the code snippet of this implementation..

```
def ensemble_recommender(customer):
    customer_purchases = t['article_id'][t['customer_id'] == customer].drop_duplicates().values
    all_recommendations = pd.DataFrame()
    for item in customer_purchases:
        coun_rec = content_based_recommender_count(item).values
        tfidf_rec = content_based_recommender_Tfidf(item).values
        cnn_rec = cnn_recommender(item).values
        temp = content_based_recommender_count(item)
        mean_val = np.mean([coun_rec[0], tfidf_rec[0], cnn_rec[0]])
        temp.iloc[0] = mean_val
        all_recommendations = pd.concat([all_recommendations, temp])
    return all_recommendations[0].nlargest(5)
```

```
def weighted_ensemble_recommender(customer):
    customer_purchases = t['article_id'][t['customer_id'] == customer].drop_duplicates().values
    all_recommendations= pd.DataFrame()
    for item in customer_purchases:
        coun_rec = content_based_recommender_count(item).values
        tfidf_rec = content_based_recommender_Tfidf(item).values
        cnn_rec = cnn_recommender(item).values
        temp = content_based_recommender_count(item)
        mean_val = np.mean([coun_rec[0] *cnn_weight ,tfidf_rec[0]*tfidf_weight,cnn_rec[0]*bv_weight])
        temp.iloc[0] = mean_val
        all_recommendations = pd.concat([all_recommendations,temp])
    return all_recommendations[0].nlargest(5)
```

Functionality of this recommendation was tested by passing random customers. The recommendations are not evaluated, the main aim was to see if the recommendation system is working.

## 6.3: Critique

This experimentation was a motivation to have a better recommendation system and having the idea of using weights was to see if we can do even better, again the evaluation chapter covers this. The downside of an ensemble approach is that it requires calling a recommendation system by giving it an input, having multiple recommendation systems in an ensemble can increase the computation time and also require the fully functional and implemented recommendation system techniques. Since the ensemble is a combination of multiple algorithms, debugging can be quite difficult and time consuming. On the other hand the content based recommendation system explored in chapter 4 still is easier to implement and is more simplistic than an ensemble based recommendation system.

All of the techniques we have explored are recommending items to a Customers to narrow down the choices they have to make. However we need to examine which strategy is best at achieving this. The next Chapter evaluates the recommendation systems and compares them through experiments. This shows us how good are the recommendations or in other words if they are relevant to the user.



# Chapter 7 : Evaluation

## 7.1: Experimentation

There are quite a few Recommendations System implemented in this project. The question that stands is how accurate these recommendation systems are? . Did a Customer purchase any of the recommended items?. How relevant are their recommendations to a Customer?. The main goal of this project is to implement and experiment with the recommendations systems in terms of their precision. The Recommendation System with highest precision is the winner. The evaluation is carried out on a test set. That has data that those recommendation systems have not seen before.

The test set contained 1500 users in total. Before the testset was created the transaction dataset was altered by only allowing the customers that had at least 5 transactions. From this transaction dataset 1500 customers were chosen at random. For each of those 1500 customers the last 20% of their transactions were added to the test set and removed from the original transaction dataset. When all 1500 random customers were taken into account the test transactions data was stored as a Csv (Comma Separated Value) file. So it can be used later to test a Recommendation System. The Recommendation systems were tested by obtaining 6 recommendations from each of the Recommender Systems discussed in all the chapters above. Precision of each of the recommenders was computed based on how many of the customer's recommendations are in the test set for that customer. This was done for all 1500 customers and then the mean of all the precisions were calculated and this was the overall accuracy of the recommendation system. Precision can be defined as,

$$\text{Precision} = \frac{\text{The number of recommendations that are relevant (User bought them)}}{\text{Total number of Recommendations}}$$

During the experimentation it was discovered that some of the recommendations had a similarity score of 1, this score implies an identical item. These scores of 1.0 can be found in the diagonal of the cosine similarity matrix representing identical items. Every recommendation system takes in a customer as input and recommends based on what that customer has bought, except the popularity and random recommendation system. The recommendations do not include the actual item that the user has bought. Hence there has to be another reason why these similarity scores of 1 were evident. This only applies to the content based recommendation systems covered in detail in chapter 4. The TF-IDF (Term Frequency- Inverse Document Frequency) and Binary Vectorization based recommendation systems take the description of the products into account.

Those descriptions were found to be identical for the items with different ids. Visually they are the same items but with different colors. This led to evaluating the content based recommenders in two Phases. Phase 1 where those duplicates are allowed and phase 2 where they are not allowed. Both approaches led to different results. Below is the visual representation of this case. The identical scoring can lead to optimistic results. Possible solution to identical descriptions is covered in the Future Work section of the final chapter. Below is the visual representation of this problem.

## Identical Scores:

```
print(content_based_recommender_Tfid('65cb62c794232651e2ac711faa11c2b4e3d41d5f3b59b50bee3ffde1d5776644'))
```

```
897901002    1.0
897901001    1.0
792753002    1.0
890866001    1.0
852181003    1.0
872813001    1.0
Name: 0, dtype: float64
```

```
print(content_based_recommender_count('65cb62c794232651e2ac711faa11c2b4e3d41d5f3b59b50bee3ffde1d5776644'))
```

```
755876004    1.0
755876002    1.0
755876005    1.0
859400005    1.0
859400004    1.0
859400007    1.0
Name: 0, dtype: float64
```

## Duplicate Descriptions

```
print(i['detail_desc'][i['article_id'] == 872813001])
print(i['detail_desc'][i['article_id'] == 872813003])
```

```
2    Short dress in an Oxford cotton weave with not...
Name: detail_desc, dtype: object
1    Short dress in an Oxford cotton weave with not...
Name: detail_desc, dtype: object
```

## The Products

```
x = [872813001, 872813003]
show_image(x)
```



## 7.2: Random and Popularity Recommendation System

Recommendation System	Precision (Mean)
Popularity Recommender	0.004
Random Recommender	0.00013333333333333334

The random recommender is the worst recommendation system in contrast with all the other recommendation systems covered in this project. However the Popularity Recommendation System is slightly better than the Random Recommender and also handles the cold start problem (where a fresh user comes in with no prior data available) See Chapter 3. Since we have a gigantic dataset available with lots of information that can be used to make relevant recommendations to customers. The Content based recommendation approach takes the users interaction into consideration and hence is better than the random and popularity recommendation systems.

## 7.3: Content Based Recommendation System

**Phase 1-** Duplicate product descriptions allowed.

Recommendation System	Precision (Mean)
Binary Vectorization Based	0.054911111111111105
TF-IDF Based	0.031488888888888889

**Phase 2 -** Duplicate product descriptions **NOT** allowed.

Recommendation System	Precision (Mean)
Binary Vectorization Based	0.047888888888888884
TF-IDF Based	0.026666666666666667

A drop in precision can be noted in both approaches. The Binary Vectorization based approach appears to have better accuracy than the TF-IDF approach. They are both vectorization based designs and take the product descriptions into account. However the design of the Binary Vectorization approach is a step ahead of the TF-IDF approach as it captures the user item interaction in binary values see( Chapter 4). Our next technique involved the use of the image dataset we had to train a Convolutional Neural

Networks Activations based Recommendation System. This system involves a lot of time in training the Convolutional Neural Network and therefore it is the most sophisticated of all the other recommendation systems implemented.

## 7.4: Convolutional Neural Network's Activations Based Recommendation System

Recommendation System	Precision (Mean)
Convolutional Neural Network Activation Based	0.20833333333333334

This Recommendation System is the winning recommendation system as it has the highest accuracy in comparison with all the other recommender systems. However the one downside is that it has a considerable amount of time spent in training the Fashion Classifier and hardware resources limitation was a great challenge. This Recommendation system is expected to have even better accuracy if the new model could have been trained on all 126,000 images and further time was spent on improving the accuracy of the multi class Fashion Image Classifier. This Recommendation system is discussed in a great detail in chapter 5 above. But Can we improve the accuracy ? The answer to this question is based on our experimentation that involves implementing an ensemble based recommendation system that takes into account the combination of all the recommendation techniques aforementioned.

## 7.5: Ensemble Based Recommendation System

Since Ensemble involved the use of TF-IDF(Term Frequency - Inverse Document Frequency ) and Binary Vectorization Based Recommendation System it was also evaluated in two phases.

**Phase 1-** Duplicate product descriptions allowed

Recommendation System	Precision (Mean)
Ensemble	0.009822222222222222

**Phase 2-** Duplicate product descriptions **NOT** allowed.

Recommendation System	Precision (Mean)
Ensemble V2	0.007766666666666666

This ensemble motivation had high expectations, as the combination of the recommendation systems techniques and averaging of the scores could very likely lead to a better accuracy than the Convolutional Neural Network based recommendation system and content based recommender systems. This experiment was carried out in order to see if we can beat our winning Convolutional Neural Network's Activations based recommender. However these surprising results made the ensemble fall around the random and popularity based recommender accuracy area.

Another interesting experimentation was carried out in an attempt to improve the accuracy of the ensemble recommendation system. This technique involved allocation of weight to the scores of the underlying recommendation systems. The convolutional neural network's activations based recommendation system was given the highest weight of **0.8**, and the binary vectorization had the weight of **0.6** and finally the tf-idf recommender had the weight of **0.3**. The scores of the recommendation made by those recommenders were multiplied by the weight. The weights were tweaked every time throughout the experimentation and above are the best weights found. The table below shows the evaluation weighted ensemble.

Recommendation System	Precision (Mean)
Weighted Ensemble	0.010244444444444445

We were able to improve the ensemble recommendation slightly with this extra effort. However this was not sufficient enough to beat the winning Convolutional Neural Network activation based recommendation system. In comparison to the original ensemble there is no difference other than the multiplying the scores of the underlying recommendation systems's weight. The motivation and idea of the weighted ensemble is same and comparison to other recommendation systems is identical as what was mentioned in the ensemble approach above.

## 7.6: Visual Evaluations.

In light of all the experimentation carried out above our best recommendation system is the Convolutional Neural Network's Activations based recommendation system. The Binary Vectorization based recommender came in at second place. Below are a few visual evaluations. To see how well these recommendations systems perform. The evaluation was done by taking one random user out of the test set. Pass this customer as input to both of the recommendation systems and obtain 6 recommendations from each recommender. Note that when the recommendations were made the **test set transactions were not looked at**. The test set transactions are solely for testing the accuracy of a recommendations system through evaluation in terms of how relevant the recommendations are and if the user actually buys them. The randomly chosen customer details are given below.

```
ID = '8db63da231ee901d7b7b1528a5e45ae52bc5fb86cf9104fbc18521bba8176763'
p = test_t[test_t['customer_id'] == ID]['article_id'].drop_duplicates().values
print(f'Total number of transactions for this customer in the test set: {len(p)}')
```

Total number of transactions for this customer in the test set: 7

Below are the items in the customer 's(mentioned above) transactions in the testset

```
show_image(u_purchases)
```



<Figure size 640x480 with 0 Axes>

Below are the visual recommendations from Convolutional Neural Network Activations based recommender

```
show_image(cnn_rec)
```



<Figure size 640x480 with 0 Axes>

The visual recommendations from the Binary Vectorization Based recommendation system are as follows.

```
: show_image(bv_rec)
```



<Figure size 640x480 with 0 Axes>

Visually these recommendations can be seen to be similar, and also the recommended items are in the customer transactions in the test set. This means that our best recommendation systems are recommending relevant items. However the question that stands is, How good are those recommendations ? Consider the table below capturing the accuracy of our 2 best recommender systems for this one customer mentioned above.

Recommendation System	Precision (Mean)
Convolutional Neural Network Activation Based	0.50
Binary Vectorization Based	0.16666666666666666

Despite the complex design and computational time in contrast with the Binary Vectorization based recommendations system. The Convolutional Neural Network Activation Based recommender system turns out to be the winner for this specific customer as well.



# Chapter 8: Conclusion and Future Work

## 8.1: Conclusion

In the project we were keen to explore different recommendation system techniques and see how they differ from each other and why one can be better than the other, this was done by evaluating each of the techniques and comparing them and finally finding our winning recommendation system. The results of the evaluation in terms of how accurate the recommendation systems are led us to conclude that our winning recommendation system was the Convolutional Neural Network's Activation based recommender. In our experimentation in chapter 7 it was evident that despite its sophisticated design and the computational time cost involved it was still proven to have best precision. The visual evaluations of this recommendation system also points to its accuracy.

However there were a range of other recommendations strategies used in our experimentation each with a motivation to be better than the previous one. We have started from the most basic form of recommendation systems such as Popularity and Random Recommenders, then one step up to content based recommendation systems, which, in contrast with basic recommendation systems, were way better as they took information from the dataset we have into account. But the challenges that were encountered during their implementation lead us to experiment recommendation system strategy based on a model that can visually classify fashion images. This was our winning Recommender, described in the paragraph above. Our final intention was to see if we can do even better by combining our already implemented techniques into an Ensemble, Through the experimentation it was realized that the Convolutional Neural Network's Activation based recommendation systems was still more accurate in comparison with the Ensemble, Finally we made one subtle change in our Ensemble technique by giving the recommendation scores weights, That significantly improved the weighted ensemble performance in contrast with The initial Ensemble technique but was not better enough to beat our image based recommendation system.

## 8.2: Future Work

In our project we have used a one month sample from H and M (Fashion Retailer) full dataset that contained the transactions that took place over the period of two years (See Chapter 2). Due to limited hardware resources we were unable to handle such a large amount of data. This was one of the main challenges that we faced during this project. If we were able to utilize the entire available data, this would have led to having more user transactions in the test-set as we used only 1500 users in our test set (see chapter 7)

and hence it would be interesting to see how well our recommendation systems perform there might have been a different winner with better precision.

Due to time constraints we were not able to do more experimentations to overcome the duplicate product description problem highlighted in our content based recommendations systems described in chapter 4 above. Since the visual representation of this problem shown in Chapter 7 shows us that the reason we have the identical description is the slight variation in the image eg two trousers can have different colors but they are both trousers. One possible solution to this problem is to go through each product description and append the color in their description. Although this can be time consuming it is a one time job. This can lead to the descriptions to be less identical and hence will lead to different scoring in cosine similarity matrix. Therefore the accuracy of Binary Vectorization Based and TF-IDF (Term Frequency - Inverse Document Frequency) based recommendation systems can improve significantly.

In our winning Convolutional Neural Network's Activation Based Recommendation System, if our Convolutional Neural Network's accuracy can be improved through more fine tuning, it can also lead to better performance of the recommendation system. However, the large dataset and 130 classes with limited hardware resources proved it to be a difficult task. But if a Pretrained Fashion Classifier can be found, that would have been extremely beneficial as it could have been used through a transfer learning approach with some fine tuning. There were a range of pre-trained neural networks available but it was quite difficult to find the one trained on actual fashion based images.

Further feature works can include the diversity of fashion based recommendations for example recommending relevant stuff can trap the user within his previous interaction with the fashion retailer, the consumer might not be happy with the purchase and every time seeing the similar items can annoy the user. The diversity in recommendation systems can overcome this problem by comparing the recommendations with each other and move the recommendations that are too similar with the ones that are quite different from what has been recommended. This re-ranking of recommendations can help the user to try some other fashion based products

Recommendation Systems is a large research area in general and is one of the popular topics in modern day AI. A variety of interactions with a recommendation system is being researched. For example A fashion recommendation system recommends a product and allows the user to interact with it, or in other words critique its recommendation. Do you have something similar but at a lower price? Like a simple conversation between a salesman and a customer. This interaction based approach points towards a conversational recommender system. Implementation of this type of recommendation system can not only be accurate but also well able to shape buyer's

purchase decisions while being more precise in relevant recommendations. This kind of recommendation system could be highly beneficial to online fashion retailers.

Derek Bridge refers "*Beyond beyond-accuracy*" is a wider set of criteria, an idea that goes beyond evaluating a quality of a recommendation system on on basis of their accuracy but also their ability to explain their recommendation, their privacy, robustness and their fairness to the users along with free from selection and systematic bias and also making responsible recommendations [14]. Fashion recommendation systems are far from this wider set of criteria in fact the modern AI is very much reliant on Human Feedback and research this implies that there is much work to be done in Modern Day AI. If there the research and development lead to a design and Implementation of a Fashion Recommender System takes take some of these wide set of criterias along and is conversational could lead to huge change fashion industry, by bringing more business and consumer attention to try these amazing and fascinating Artificially Intelligent technologies in modern day Computer Science.

# Bibliography

- [1] Sharma, Lalita, and Anju Gera. 2013. "A Survey of Recommendation System: Research Challenges." *International Journal of Engineering Trends and Technology (IJETT)* 4, no. 5 (May): 1989-1992.
- [2] Bridge, Derek. 2023. *CS4619: Artificial Intelligence, Recommender Systems I*. N.p.: Derek Bridge.  
[https://github.com/derekgbridge/artificial\\_intelligence/blob/master/ai2/lectures/AI2\\_06\\_recommender\\_systems\\_1.ipynb](https://github.com/derekgbridge/artificial_intelligence/blob/master/ai2/lectures/AI2_06_recommender_systems_1.ipynb).
- [3] H&M Group. 2022. "H&M Personalized Fashion Recommendations." Kaggle.  
<https://www.kaggle.com/competitions/h-and-m-personalized-fashion-recommendations/>.
- [4] Bridge, Derek. 2023. *CS4619: Artificial Intelligence II - Bag of Words*. N.p.: Derek Bridge.  
[https://github.com/derekgbridge/artificial\\_intelligence/blob/master/ai2/lectures/AI2\\_01\\_bag\\_of\\_words.ipynb](https://github.com/derekgbridge/artificial_intelligence/blob/master/ai2/lectures/AI2_01_bag_of_words.ipynb).
- [5] Le, James. 2018. "CNN with 3 Convolution Layers." Github.  
<https://github.com/khanhnamle1994/fashion-mnist/blob/master/CNN-3Conv.ipynb>

- [6] Bridge, Derek. 2022.** *CS4168: Artificial Intelligence I - Convolutional Neural Networks*. N.p.: Derek Bridge.  
[https://github.com/derekgbridge/artificial\\_intelligence/blob/master/ai1/lectures/AI1\\_21\\_convnets.ipynb](https://github.com/derekgbridge/artificial_intelligence/blob/master/ai1/lectures/AI1_21_convnets.ipynb).
- [7] K V, Greeshma. 2021.** “Fashion-MNIST Dataset Images with Labels and Description II.” Research gate.  
[https://www.researchgate.net/figure/Fashion-MNIST-Dataset-Images-with-Labels-and-Description-II-LITERATURE-REVIEW-In-image\\_fig1\\_340299295](https://www.researchgate.net/figure/Fashion-MNIST-Dataset-Images-with-Labels-and-Description-II-LITERATURE-REVIEW-In-image_fig1_340299295).
- [8] Bridge, Derek. 2022.** *CS4618: Artificial Intelligence I - Neural Networks*. N.p.: Derek Bridge.  
[https://github.com/derekgbridge/artificial\\_intelligence/blob/master/ai1/lectures/AI1\\_17\\_neural\\_networks.ipynb](https://github.com/derekgbridge/artificial_intelligence/blob/master/ai1/lectures/AI1_17_neural_networks.ipynb).
- [9] Bridge, Derek. 2022.** *CS4618: Artificial Intelligence I : Logistic Regression*. N.p.: Derek Bridge.  
[https://github.com/derekgbridge/artificial\\_intelligence/blob/master/ai1/lectures/AI1\\_14\\_logistic\\_regression.ipynb](https://github.com/derekgbridge/artificial_intelligence/blob/master/ai1/lectures/AI1_14_logistic_regression.ipynb).
- [10] Xiao, Han, Kashif Rasul, and Roland Vollgraf. 2022.** “fashion\_mnist.” TensorFlow. [https://www.tensorflow.org/datasets/catalog/fashion\\_mnist](https://www.tensorflow.org/datasets/catalog/fashion_mnist).

**[11] Bridge, Derek. 2022.** *CS:4618: Artificial Intelligence I- Training a Convolutional Neural Network* n. N.p.: Derek Bridge.

[https://github.com/derekgbridge/artificial\\_intelligence/blob/master/ai1/lectures/AI1\\_22\\_training\\_a\\_convnet.ipynb](https://github.com/derekgbridge/artificial_intelligence/blob/master/ai1/lectures/AI1_22_training_a_convnet.ipynb).

**[12] Makwana, Aakash. 2022.** “Recommending Similar Images using Image Embedding -.” Analytics Vidhya.

<https://www.analyticsvidhya.com/blog/2022/07/recommending-similar-images-using-image-embedding/>.

**[13] Bridge, Derek. 2022.** *CS4618: Artificial Intelligence I- Decision Trees and Random Forests*. N.p.: Derek Bridge.

[https://github.com/derekgbridge/artificial\\_intelligence/blob/master/ai1/lectures/AI1\\_15\\_decision\\_trees.ipynb](https://github.com/derekgbridge/artificial_intelligence/blob/master/ai1/lectures/AI1_15_decision_trees.ipynb).

**[14] Bridge, Derek. 2023.** *AI2 -Recommender Systems IV*.

[https://github.com/derekgbridge/artificial\\_intelligence/blob/master/ai2/lectures/AI2\\_18\\_conclusions.ipynb](https://github.com/derekgbridge/artificial_intelligence/blob/master/ai2/lectures/AI2_18_conclusions.ipynb)

**[15] Saket, Sheel. 2020.** “Count Vectorizer vs TFIDF Vectorizer | Natural Language Processing.”

<https://www.linkedin.com/pulse/count-vectorizers-vs-tfidf-natural-language-processing-sheel-saket/>.



