

# CHAPTER-1

## INTRODUCTION

### 1.1 Overview

Recommendation engines are able to spot data patterns in the data set by analysing client preferences and generating out suggestions that are pertinent to their requirements and interests. In real-world applications like Amazon, a recommendation engine has been used to propose products that buyers would like. Recommender systems have grown in importance as a result of the emergence of web services like YouTube, Amazon, Netflix, and many more over the past couple of decades. Whether it is in e-commerce (which suggests to customers articles that may interest them) or online advertising, recommender systems are becoming an essential component of our daily online activity (suggest to users the proper contents, matching their tastes). Content-based filtering is a common recommender design technique. In content-based filtering approaches, the item description and user preference profile are important variables. When an object already has known information (such as a name, location, description, etc.), but not the user, these strategies perform well. Content-based recommendation systems recognise a user's preferences in light of an item's qualities and approach suggestions as a separate user issue. The algorithms try to recommend goods that are similar to the user's past or present preferences. This temporary profile is created without the usage of a user login device. A number of candidate items are specifically compared to goods that have garnered positive user reviews. This method is based on research on information retrieval and data filtering.

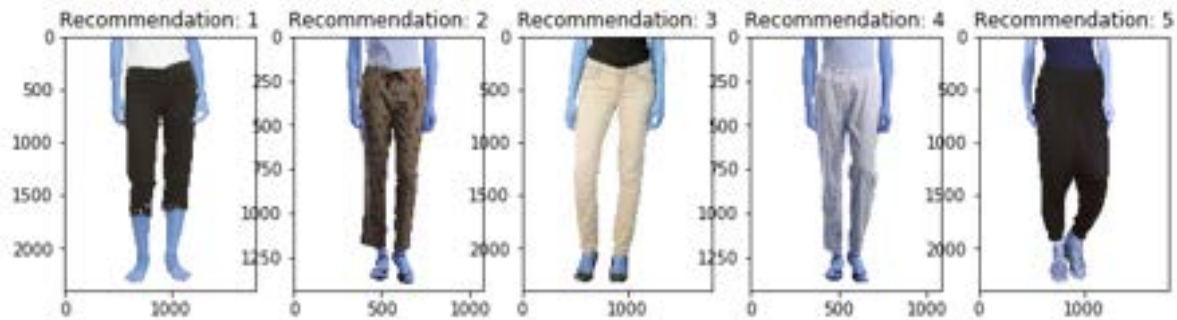


Fig 1.1

## 1.2 Objective

Smart apparel recommendation system powered by deep learning. One feed-forward neural network is developed as the recommender, and two inceptions-based convolutional neural networks are created as the prediction component to recommend a fabric. In this study, we reach accuracy rates of 98 percent for colour prediction, 86 percent for gender and fabric pattern prediction, and 75 percent for garment selection. Deep fashion recommendation system style feature decomposition. Due to the jumbled information of style and category, the clothes vector, however, frequently suggests mismatched clothing.

### Traditional RS

- content-based and collaborative filtering (CF) systems.
- recommendations on historical interactions and user/item attributes.
- cold start problem is an issue of irrelevant recommendations for a new user who still has performed few system interactions.

### Deep learning RS

- NCF + Tensorflow Transfer learning + other DNNs.
- good fit for unstructured multimedia data processing given effective feature extraction.
- CNNs help to eliminate the cold start problem like collaborative filtering.




Fig 1.2

To solve this problem, we propose a style feature extraction (SFE) layer that divides the apparel vector into styles and categories. To obtain more exact style information, we extract and remove the category metadata from the clothing vector. The category information is the collection of features that slightly vary within the same class based on the attributes while remaining different from one another.

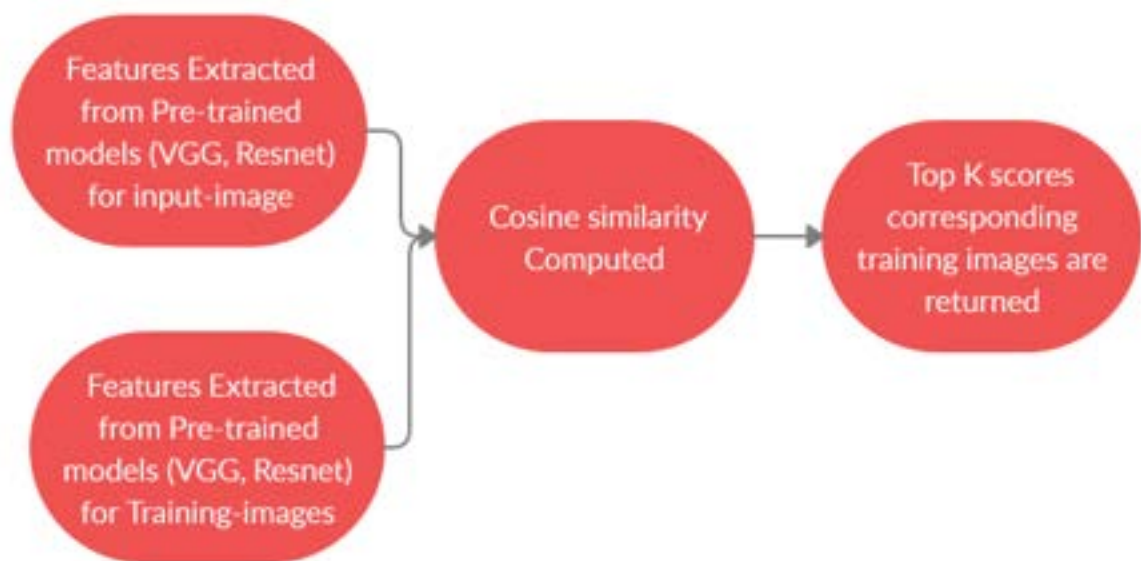


Fig 1.3

### 1.3 Goals of the project

The project's objective is to develop a model that can offer fashion advice just by looking at a picture of itself. When a model agrees to participate in a picture, she first judges whether or not it includes a piece of fashion before recommending it.

This study's main objective is to:

- Develop a fashion suggestion system that offers solutions to inquiries concerning apparel buying.

- To identify the fashion subcategory of the input image provided.
- If the fashion illustration offered is accurate, a matching outfit of clothing will be recommended.

- Purchasing goods from several e-commerce sites that were located using similar search phrases.

## 1.4 Design Workflow

1. The project uses ResNet50, a specialised CNN architecture, for picture classification. Here is the flow, which summarises the methodology:
2. employing a convolution layer to filter while extracting features.
3. employing pixel shifting within the input matrix to advance.
4. CNN kernel padding an image as it is processed.
5. ReLu is the activation function.
6. The matrix is then flattened into computed 0,1s and loaded into the neural network layer.

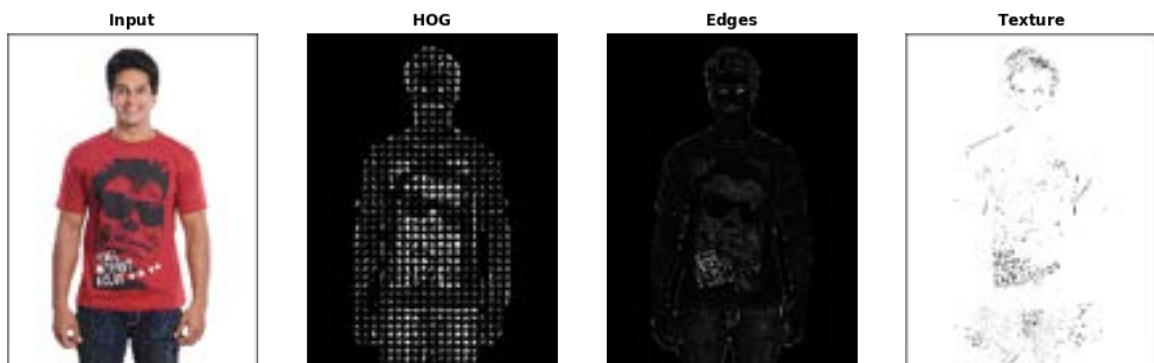


Fig 1.4

## Digging Deeper with Deep Learning

Therefore, in an educated guess, the project lead to observe/investigate techniques that might optimise us to extract more descriptive and distinctive

feature representation for the images, which can then be used in computing similarity score on the basis of which top K recommendations can be given to the user. Deep learning techniques, which are very good at spotting patterns and features in images, came into focus in our search for such feature representation.

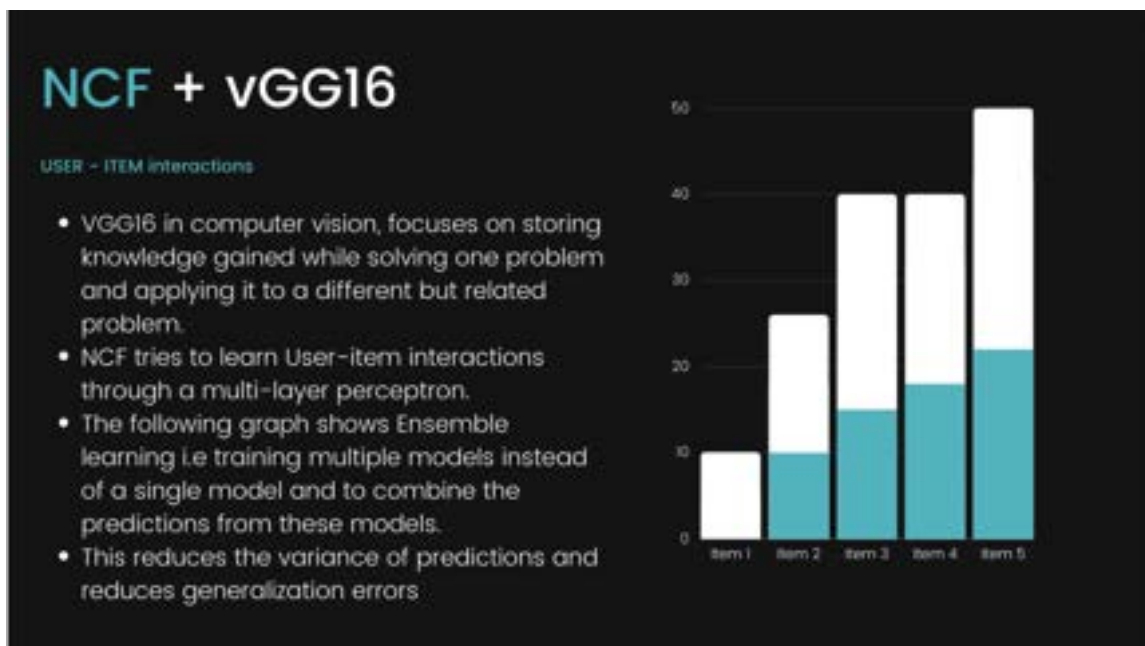


Fig 1.5

## 1.5 Database

The database should store details such as:

### 1) Product\_data

- product\_name
- product\_id
- product\_images

### 2) product\_metadata

- product\_image\_tags
- Similarity indexes

- Created\_at & modified\_at

### 3) User\_metadata

- User\_name
- user\_id
- user\_sessions

## **CHAPTER \* BASELINE ALGORITHMS AND RELATED WORK**

In ecommerce platforms, the recommender system is frequently used. The primary goal of conventional algorithms is to produce customised top-k item lists for consumers. The top-k recommendations, however, only apply to some instances in real-world platforms. For instance, only the personalised top-k recommendations can be included in the homepage recommendation on our e-commerce platform, and only 4% of user visits originate from the site. The recommendation of related products, which accounts for 33% of user visits to our platform, is the subject of this study. On e-commerce platforms, similar product recommendations are frequently used in a variety of settings. When a user visits an item, a list of comparable products will be shown to them so they may compare them to other products and make the best decision possible.

Such algorithms are used in general recommendation scenarios, such as personalised recommendations based on user behaviours, although they are created for similarity computation. The similar product lists are not personalised, the algorithms cannot process user real-time activities and generate the most recent interests, they are rigid when dealing with special requirements, and item similarity and user interests are not combined. Without taking the recommendation scenario into account. Instead of just calculating the similarities between items, our model will take into account many important aspects of real-world recommender systems, including real-time user interests, balancing the influence of user interests and item similarity, time efficiency, and other business requirements. Similar products calculations are different from personalised recommendations in that they are more concerned with the current item than with user preferences.

Most algorithms will only take user behaviour into account when calculating how similar goods are to one another (Linden, Smith, and York 2003). It makes sense in certain respects because the item the users are browsing is directly tied to similar product recommendations, which heavily emphasise the users' current interests. Additionally, rather than being limited to a small selection of items produced by personalised recommendation algorithms, a non-personalized recommendation can assist users in discovering new and different items. However, by presenting the items that are best matched with user interests, personalised recommendation algorithms can significantly enhance the user experience.

According to our dataset, 33% of all user clicks originate from non-personalized recommendations for similar products, however only 25% of these clicks result in add-cart occurrences. The conversion rate from click to add-cart is substantially lower when compared to our tailored recommendation on the homepage (4% click generates 6% add-cart). Therefore, for our system, a customised approach to similar product recommendations is required. A personalised model must also process in-the-moment user activities because the user's present interests have a significant impact on this kind of suggestion situation. In order to create a unique algorithm here,

We face the following difficulties:

- It is important to take into account both users' recent and past browsing patterns. The algorithm must strike a balance between user interests and the influence of the current content.
- The algorithm must take into account user behaviour in real-time. We must use the most recent user visit history to construct user interests vector since similar product recommendations are greatly influenced by short-term user interests.
- More things may be displayed to users in the suggestion results.

In the field of natural language processing, deep neural networks like RNN/LSTM, which capture the innate sequential structures of data, have seen encouraging success. Applying such sequential neural networks to recommendation tasks to mine the temporal dynamic characteristics of user behaviour is flexible and intuitive. suggests a recommendation mechanism that is session-based. The model generates the likelihood for each potential item in the session using the one-hot sequential encoding as input. The

proposed RNN-based model, in contrast to the previous session-based approach without learning user representation, is capable of simulating changes in user interests and item features over time. In recent years, tailored neural network recommendations have become very popular. The authors of show how the YouTube recommender system works.

The multilayer perceptron receives input from embedded videos and other elements. (Zhou et al. 2018) suggest using a neural network model to manage various features and the proximity of items based on the attention model. Many customised recommendation algorithms also introduce the wide and deep framework (Cheng et al. 2016). In this study, we construct a neural network model to produce real-time user interests. The model can then offer a tailored recommendation for a similar product based on the candidate pool. Standard Algorithms The three baseline algorithms we suggest in this section are an item-to-item collaborative filtering algorithm, an image-similarity-based algorithm, and the item2vec algorithm. We will initially create a candidate pool for similar product recommendations, subject to our online business regulations. There are three different sources for the items in the candidate pool: products with similar attributes, similar new products, and products produced by algorithms. Each item has a pool of 200 comparable items. We score the products in the pool during the ranking stage and present the top 30 things to buyers. We ignore the specifics of creating the pool and concentrate on the algorithms for ranking.

## **CHAPTER-2**



# MATERIALS AND METHODS

## 2.1 PyTorch

There is no dataset or task more well-known in the field of picture categorization than ImageNet. The objective of ImageNet is to precisely categorise input photographs into a set of 1,000 common object categories that real-world computer vision systems "see."

Pre-trained networks are a feature of the majority of well-liked deep learning frameworks, including PyTorch, Keras, TensorFlow, fast.ai, and others. Researchers in computer vision trained these cutting-edge models on the ImageNet dataset, and they are extremely accurate.

After finishing their training on ImageNet, researchers stored their models to disc and then made them available for free download by other researchers, students, and developers to use in their own work.

The following cutting-edge classification networks will be demonstrated in this tutorial's use of PyTorch to categorise input images:

- VGG16
- VGG19
- Inception
- DenseNet

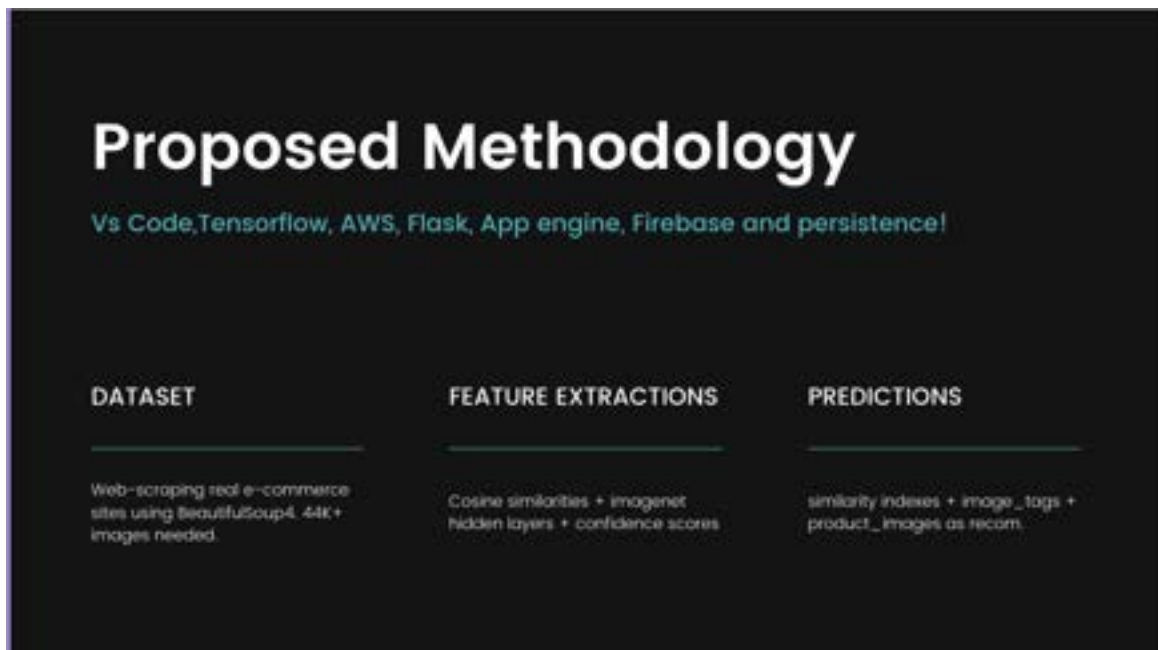


Fig 2.1

When you try to identify an object in an image, you start to take note of characteristics like colour, shape, and size that aid in object recognition. The CNN use the same method. The convolution and pooling layer, where the model records the features in the image, and the fully connected (FC) layer, where classification occurs, are the two main layers of a CNN.

Visualizing the workflow:

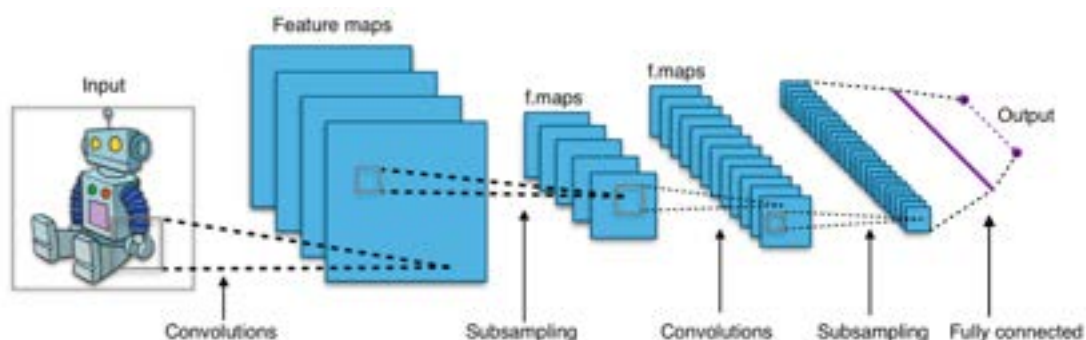


Fig 2.2

## **2.2 Tensorflow**

Google's open source machine learning framework TensorFlow is utilised for a range of tasks. The nodes of the graph represent mathematical operations, while the edges represent the multidimensional data arrays that are transmitted between them.

Using labelled sample photos, a model is trained to detect the target classes (objects to identify in images). In a word, the problem of image categorization is one of supervised learning. Raw pixel data was the only input for early computer vision algorithms. However, as seen in Figure 2, plain pixel data by itself is inadequate to sufficiently characterise an object's diverse changes.

## **2.3 NumPy**

- To work with arrays, utilise the NumPy Python module.
- Matrix operations, the Fourier transform, and functions for working with linear algebra are also included.
- NumPy was created by Travis Oliphant in the year 2005. It is an open source project, therefore using it is free.
- NumPy is the abbreviation for numerical Python.

## **2.4 Cosine Similarity**

The cosine similarity index determines how similar two vectors are to one another in an inner product space. Consequently, the cosine features of such processed images are essentially the angle formed by two vectors pointing in the same direction. It is commonly used in text analysis to determine how similar two documents are.

The frequency of a certain word or phrase (such as a keyword) inside the text is tracked by each of the thousands of attributes that can be found in a document. A term-frequency vector can therefore be used to represent any document as an object. For instance, Table 2.5 demonstrates that while the term "hockey" only appears three times in Document 1, the phrase "team" appears five times. Coach is absent from the entire work, as can be seen. by a word coach is absent from the entire document, as indicated by a count value of 0. Such data can be highly asymmetric.

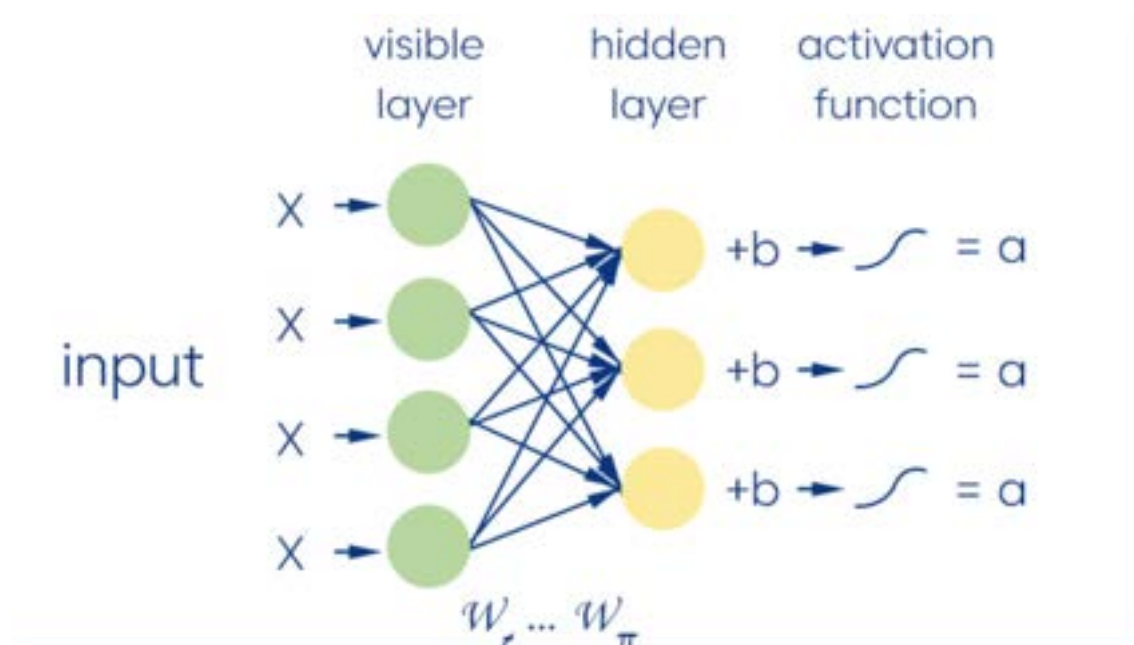


Fig 2.3

## 2.5 Azure ML

The data, method, and trained model that make up a machine learning solution are all, or in part, given by a cognitive service, according to AZURE Cognitive Services. These services are made to be utilised by anyone with no previous machine learning or data science experience and simply a general understanding of data. These services provide both REST APIs and language-based SDKs. As a result, you need to be conversant with programming languages in order to use the services.

## **2.6 GCP VIDEO AI**

### AutoML Video Intelligence

The graphical interface of AutoML Video Intelligence makes it straightforward to train your own special models to recognise and track things in movies even if you have no prior experience with machine learning. It's ideal for tasks that the pre-trained Video Intelligence API can't complete since they require distinctive labelling.

### Video Intelligence API

The Video Intelligence API can automatically detect a variety of objects, locations, and behaviours in both recorded and streaming video with the aid of pre-trained machine learning models. It is of excellent quality straight out of the box, works well in most situations, and gets better with time as new concepts are added.

## **2.7. Streamlit**

Using the free, open-source Streamlit framework, data scientists can quickly construct interactive dashboards and machine learning web apps without any prior experience of front-end web programming. If you are all proficient with Python, you can all utilise Streamlit to create and share your web apps in hours rather than weeks.

I like Tableau because of its rich analytics and visualisation capabilities, but I've been looking for an adaptable, cost-free alternative that can grow for my own analytical projects.

The instant I came upon Streamlit, I was smitten! Since everything is written in Python, using Streamlit to launch your dashboards or data apps will be a cinch if you are comfortable with Python (as many data scientists are).

It is completely written in Python, so if you are familiar with it (as many data scientists are), you can use Streamlit to quickly and easily get your dashboards or data apps up and running with just a few lines of code, without having any prior experience with front-end web application development.

## 2.8 Scikit learn

Sklearn is the name of the most trustworthy and reliable Python machine learning library (Skit-Learn). It provides a number of efficient tools for statistical modelling and machine learning, such as classification, regression, clustering, and dimensionality reduction, through a Python consistency interface. This library was mostly created in Python and is based on NumPy, SciPy, and Matplotlib.

Instead of loading, altering, and summarising the data, the Scikit-learn library focuses on modelling it. Some of the most well-known model categories that Sklearn provides are as follows:

The Supervised Learning Algorithms Scikit-learn contains almost all popular supervised learning techniques, such as Linear Regression, Support Vector Machine (SVM), Decision Tree, and others.

However, it also incorporates all of the widely used unsupervised learning techniques, such as factor analysis, clustering,

On the other hand, it also incorporates all of the widely used unsupervised learning techniques, such as unsupervised neural networks, factor analysis, PCA (Principal Component Analysis), and clustering.

Unlabeled data are clustered using this technique.

A method for assessing the accuracy of supervised models on unseen data is cross validation.

By reducing the number of attributes in data, dimensionality reduction can be used to narrow the number of options for feature selection, summarization, and visualisation.

A weighted output is produced from various supervised models via methods like ensembles, which are ensembled on top of one another.

To develop supervised models, useful qualities are found through feature selection.

Open Source The library is free and open source.

## **2.9 HTML**

HyperText Markup Language, or HTML, is the foundational language of the Internet. With its aid, users can make use of and establish the significance of the internet materials. Links connecting online pages—whether they are on the same website or separate websites—are referred to as "Hypertext." The internet relies on links to function. Users can participate and contribute to the enormous WWW by publishing their own created content online and using links to connect it to other users' provided content. HTML uses "markup" to designate media in a web browser, such as text and images. When web browsers request HTML documents from web servers or specific local storage, the web browsers render the requested HTML documents. Web



browsers request HTML documents from web servers or specific local storage, and the web browsers render the documents into multimedia pages.

Despite the fact that Berners-Lee created HTML in late 1991, "HTML 2.0" was the first published standard HTML definition. HTML 4.01, which was released in late 1999, was a significant HTML version. The most recent version of HTML is HTML-5, a development of version 4.01. The year of publishing was 2012. However, HTML 4.01 is also frequently employed. HTML has a wide range of applications, including: In order to support HTML pages on laptops, desktop computers, tablets, and mobile devices, the user interface (UI) is responsive.

HTML is a language used to construct web pages that are shown online. Web pages nearly always contain HTML tags to aid browsers in comprehending and displaying the content and characteristics of the pages. The HTML pages that have already been loaded can be accessed offline on a device without the need for internet connectivity once again. broader applications: HTML5 is promoting growth across a number of sectors and may be applied in

industries like gaming creation. Other technologies that are frequently used in the creation and development of web pages include JavaScript for defining functionality and CSS for styling the web page and displaying appearance.

## **2.9 CSS**

HTML is a language that has been used to construct web pages. Nowadays, almost every website that is accessible online has HTML elements that enable browsers to comprehend and display the material and specifics of the websites. An offline browser on a device without an internet connection can access the loaded HTML pages. Expanded uses - HTML5 is facilitating innovation across numerous fields and might be used to propel growth in industries like gaming. Another tech stack that is frequently used in web page design and development is CSS, which is used to style and depict the appearance of websites, and JavaScript, which is used to define functionality. This division can increase the accessibility of the material, provide greater freedom and control when defining presentational qualities, and permit

several web pages to share formatting, and enable the .css file to be cached to improve page load speed between the pages that share the file and its formatting. By putting the necessary CSS specifications in a separate .css file, the structure content is made simpler and less repetitious. John Wium On October 10, 1994, Lie proposed CSS. Tim Berners-Lee and Lie were both employed with CERN at the same time. Evidently, other stylesheet languages were proposed that have properties similar to CSS. CSS allows for the creation of multiple style sheets that can be applied to a web page document as "cascading" styles. Examples of CSS applications include: effective over time CSS sheets that have already been produced can be used by several HTML pages. quicker load times It is unnecessary to continuously write HTML tag attributes when CSS is used. The CSS rule for a tag only has to be written once and then used again. This results in less code being written. The result is loading times are quicker. Hank Wium Lie proposed CSS on October 10, 1994. Lie was working at CERN at the same time as Tim Berners-Lee. Evidently, other stylesheet languages with features comparable to CSS were suggested. CSS allows for the creation of multiple style sheets that can be applied to a web page document as "cascading" styles. Examples of CSS applications include: effective over time CSS sheets that have already been produced can be used by several HTML pages. Using

CSS eliminates the need to continually write HTML tag attributes, which results in faster loading speeds. The CSS rule for a tag only has to be written once and then used again. This results in less code being written. The outcome is faster loading times. All relying websites on the earth would be quickly updated if the style were to be readily changed to reflect a change that needed to be reflected globally. CSS is superior to HTML because it provides a greater variety of features than HTML, making it better for formatting web pages. When compared to HTML characteristics, CSS may provide a web page a far better appearance. Style sheets enable material to be styled for use on a variety of devices. The same HTML document can be used to produce several versions of a website that are suitable for different devices, such as phones, laptops, tablets, etc., thanks to CSS.

## **CHAPTER-3**

### **RESEARCH PUBLICATION**

## **Cross-Market Product Recommendation**

Utilizing information from related, resource-richer auxiliary markets, we investigate the issue of proposing pertinent products to users in relatively resource-constrained marketplaces. According to our hypothesis, performance in one market can be enhanced using data from another.

There haven't been many studies in this field, in part because there aren't many publicly available experimental data. In order to achieve this, we gather and make available XMarket, a sizable dataset containing 52.5 million user-item interactions and encompassing 18 local markets over 16 different product categories.

We define the issue of cross-market product suggestion, often known as market adaption. We investigate various market-adaptation strategies motivated by cutting-edge methods for meta-learning and domain adaption. and put out the FOREC new neural method to market adaptability. Pre-training, forking, and fine-tuning are the three steps our model takes to make the most of data from both the target market and an auxiliary market. We carry out comprehensive studies to investigate how market adaptation affects various market pairs.

When compared to the competitive baselines we used for our investigation, the performance on the target markets continuously improves thanks to the robust effectiveness of our recommended approach. In instance, when compared to the NMF baseline, FOREC improves nDCG@10 by an average of 24% and as much as 50%. Our research's analyses and experiments point to certain future directions. For scholarly purposes, we make our data and code available.

## **Large-scale Real-time Personalised Similar Product Recommendations**

An image-similarity-based model is what we start with. The majority of our e-commerce dataset is made up of clothing. The user's decision-making process for fashion products is heavily influenced by the style, colour, and several other factors that may be easily retrieved from the image.

Therefore, in our initial model, we attempt to produce recommendation outcomes using the similarity between item images. Here, we create a feature vector for each product image using the transfer learning method. A trained model may efficiently extract characteristics from photos by using transfer learning. In the experiment, we use the ResNet152 pretrained model from TensorFlow Keras to implement the technique.

We remove the final dense layer and use average pooling to create a 2048-length vector for each product image. Then we rank the similarity between items by the value of cosine similarity.



Fig 3.1

# Context-aware Retail Product Recommendation with Regularized Gradient Boosting

The FARFETCH Fashion Recommendations Challenge 2021 has made the problem statement available. We have received a sizable sample of impressions from the FARFETCH'd recommendations algorithm, along with any accompanying click events.

An impression is a list of 6 goods that a user has seen in a recommendations UI module in a specific situation (kind of page, date, etc.), together with a label indicating which of the 6 products they have clicked on.

The competition's objective is to forecast the best placement of the six goods in a given impression, with a lower placement for those whose chances of being clicked by the user in a recommendation context are higher.

## 2.0.1 Dataset Description :

The dataset represents a sizable sample of impressions and related click events from FARFETCH's recommendations system that were recorded over a two-month period. A query id, a special identifier of a recommendation impression, is present in each row of data. For the initial entries in the competition, we had access to a validation set of about 0.6 million rows

without a target label and a training set of 3.5 million rows with a target label, is click. The dataset includes 813729 distinct impression lists, 443150 unique products, and 229064 unique users. The attribute data for each of the products is provided in a distinct attribute dataset. Taking into account all the information, there are 3 numerical qualities and roughly 20 categorical ones. The target label identifies which of the 6 products in the impression were clicked. The position of the products within the list is unknown in this dataset. In the final phase, we had access to another unlabelled test set with around 0.6 million rows for final evaluation.

## **MODELS USED:**

### **Gradient Boosting Model**

By progressively, additively, and sequentially training several models, gradient boosting is a traditional machine learning technique for transforming weak learners into strong ones. A new tree is fitted on a modified version of the original data set in the boosting strategy. This method was used to create our classifier model.



## **Ranking Approach**

For the ranking strategy, we used LightGBM. We utilise lambdarank as an objective function and use it for ranking. The concept behind lambdarank is to substitute cost for the gradient of cost with respect to model score. Along with the characteristics and labels, a list that includes the length of each group, the number of goods in each impression, or the number of impressions corresponding to each query id, is a crucial input to the ranking model. The predicted probability values acquired from the trained LightGBM Ranker model are used to rank the products within a query.

## **Binary Classification Approach**

This method treats the task as a straightforward binary classification problem, with the aim being whether or not a product has been clicked, that is, whether click is a value of 0 or 1. The likelihood that the product will be clicked on is expected to range from 0 to 1, with a number closer to 1 indicating a higher likelihood. We have tested with the classifiers LightGBM-[4] and XGBOOST [2]. In Table 1's results, we can see that the XGBOOST model obtains a greater MRR than the LightGBM model in both the scenarios, with and without the XGBOOST. This suggests that the latter model is more successful.

# **Image Based Fashion Product Recommendation with Deep Learning**

Our two-stage deep learning architecture creates fashion image recommendations based on other input images with a similar aesthetic. A neural network classifier is employed as a data-driven, visually aware feature extractor for that reason. The latter is subsequently used as input for ranking algorithm-based recommendations that are similarity-based. On the Fashion dataset, which is available to the public, our method is tested. We describe initialization algorithms that draw on transfer learning from bigger product databases. Our approach can be used in conjunction with other established content-based recommendation systems to improve robustness and performance, for instance by better corresponding to a specific customer style.

To extract features for similarity recommendations, utilise the CNN classifier. It can be applied, for instance, to e-commerce, where buyers can upload a specific fashion image, and related products are then made available based on the texture and category properties of the provided image

by the user. It is simple to add more feature extractors, such as those trained on gender or colour categorization tasks. Additionally, expansion to other fields makes logical, such as music suggestion based on unprocessed music data, but requires additional research. Our strategy has a number of intriguing extensions that are feasible. First, integrating the two distinct training phases into a single one and offering end-to-end deep learning-based fashion product suggestions would be intriguing. Siamese networks should be taken into account in particular. Moreover, hybrid strategies combining image-based and content-based systems will be implemented. Finally, it is important to evaluate the customer impact of our image-based approach and its extensions against other recommender systems through customer survey

## **Rich-Item Recommendations for Rich-Users: Exploiting Dynamic and Static Side Information**

In this article, we examine the issue of recommendation systems where the users and the things that need to be recommended are rich data structures with a variety of entity types and side-information sources in

the form of graphs. We offer a broad formulation for the issue that generalises numerous previous formulations and captures the complexity of contemporary real-world proposals. Each person or document that needs a recommendation and each item or tag that is to be recommended are both characterised by a set of static entities and a dynamic component in our formulation. Several weighted bipartite graphs are used to represent the relationships between entities. We offer MEDRES, a novel deep-learning architecture based on multiple graph-CNN interactions, to efficiently exploit these complex interactions and develop the recommendation model.

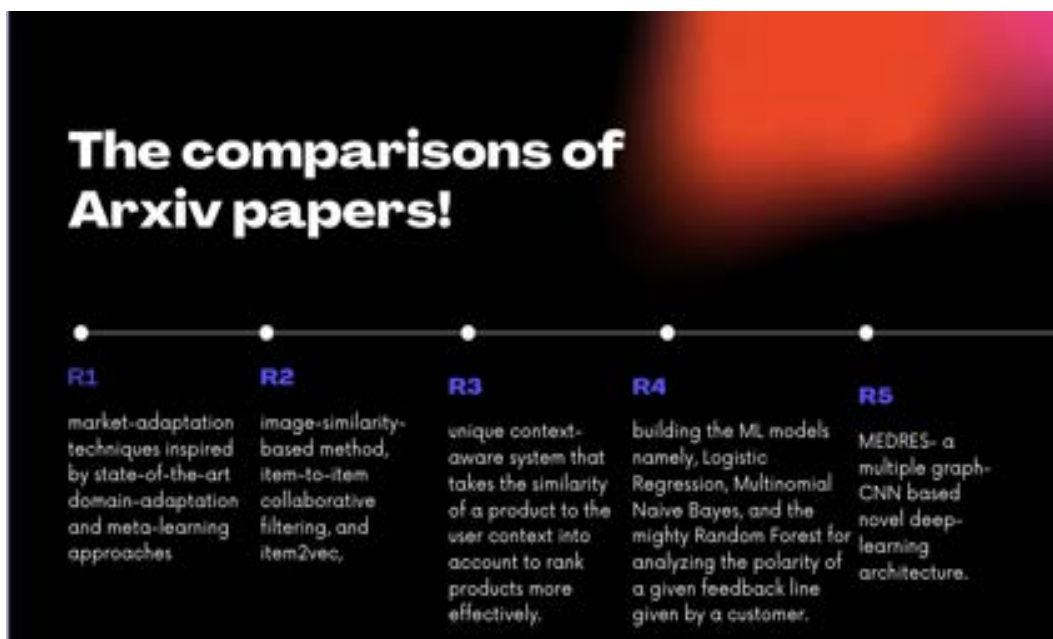


Fig 3.2

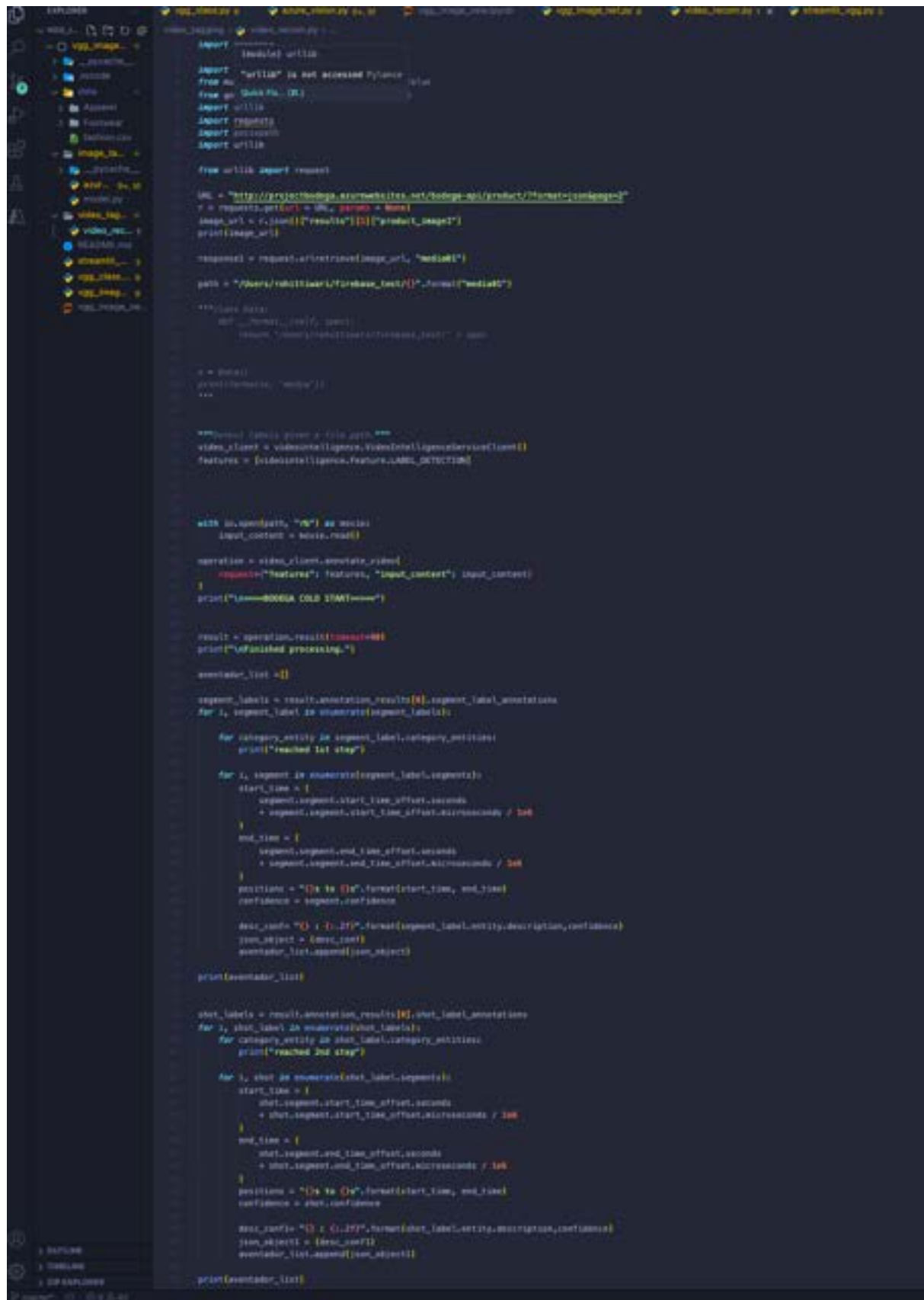
A unique graph convolution network block called AL-GCN is used by MEDRES to extract potent representative features from the underlying graphs. Moreover, we present a unique ranking metric called  $pAp@k$  coupled with a method to directly optimise the metric in order to reflect the extremely varied involvement of various users with the system and limits on the amount of items to be recommended. Using the citation

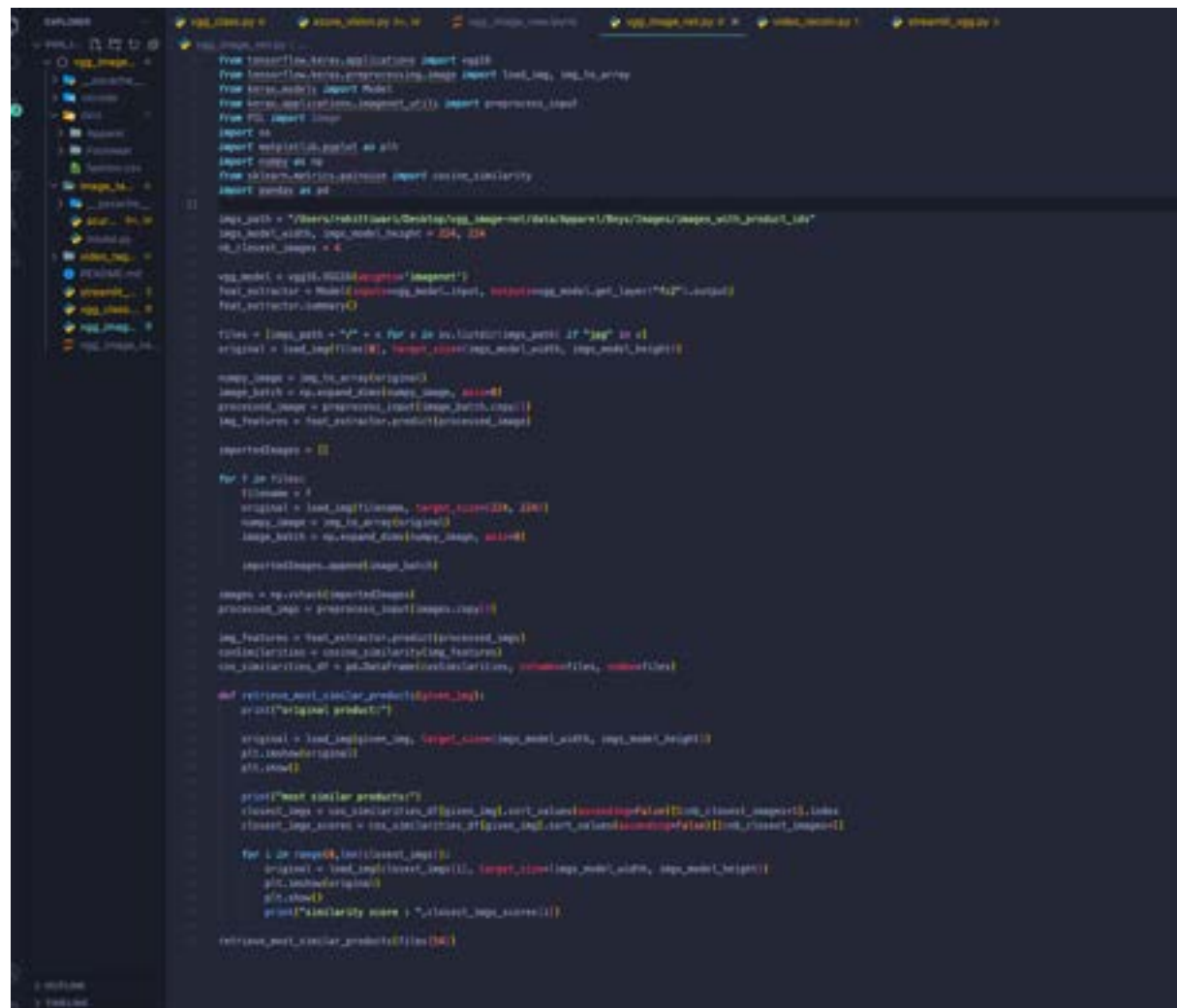
data and the Flickr data as our standards, we show the success of our methodology. We also provide two actual case studies that use both the MEDRES architecture and our formulation. We demonstrate the application of our technique to naturally model the message recommendation problem and the teams recommendation problem in the Microsoft Teams (MSTeams) product and show that it is 5-6% points more accurate than the production-grade models

For the MEDRES architecture, we presented AL-GCN, a novel GNN block that may be utilised to train graph weights and extract more potent task-specific signals. Our training procedure can make MEDRES more effective for the suggested  $pAp@k$  measure. Finally, we demonstrated the usefulness of MEDRES against various baselines by modelling a variety of recommendation challenges, including two real-world tasks, using our framework. Several other embedding techniques, such as Heterogeneous Information Networks (HIN), which may be richer in some settings than GCN and AL-GCN, can be used with our framework due to its generality. Further investigating the optimization of the  $pAp@k$  metric and examining its theoretical properties is another fascinating research area.

## SOURCE CODE

[illegible]







[illegible]



## **CHAPTER-5**

### **DRAFT RESULT AND DISCUSSION**

#### **Similarity Computation and Generating Recommendations**

We produce training vectors for each of the training and testing photographs. The cosine similarity between each test image and the cluster leaders from all 143 prepared clusters is then calculated. In this example, the leaders of each cluster are chosen at random. This is carried out for all of the various feature descriptor types that have already been covered. The mean of all cosine similarities is then calculated, and the top 5 leaders are chosen. The cosine similarity (idx features) between each image in the five chosen clusters and the test image is then calculated. For each of the numerous feature descriptor types mentioned before, this procedure is repeated. The top k images that are most alike come last.

**This low level features model's accuracy did not turn out to be very good as it could only hit the mark of 51%.**

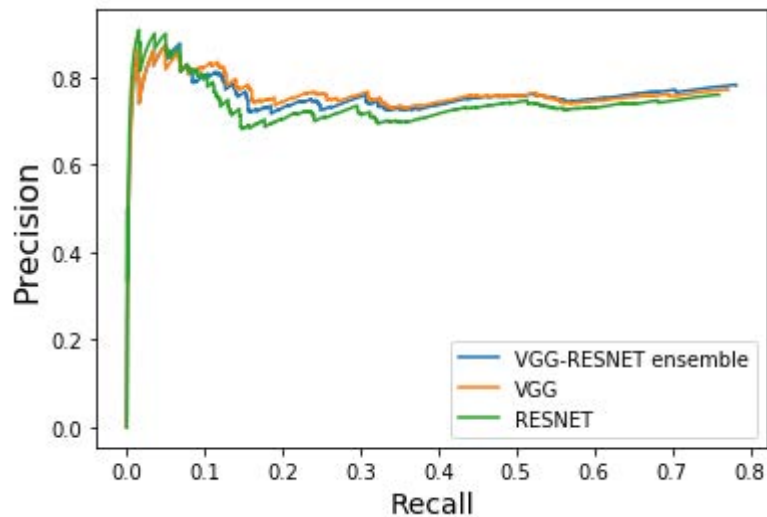
## **VGG + Resnet based Weighted Ensemble Technique**

We concluded that VGG and Resnet were the most optimised models among all of them after evaluating our models. As a result, we decided to keep refining these models to get better results.

In order to create a thorough feature representation that incorporates both the representation of VGG and Resnet models, we used input+weights, which is the weighted average of the two features. Since the size of the Resnet feature vector was larger than the size of the VGG feature vector, we used SKlearn's SelectK feature reduction strategy, which selects the top K features from a collection of features based on the goal value for the features. we used SKlearn's SelectK feature reduction technique, which selects the top K features from a collection of features depending on the goal value for the features.

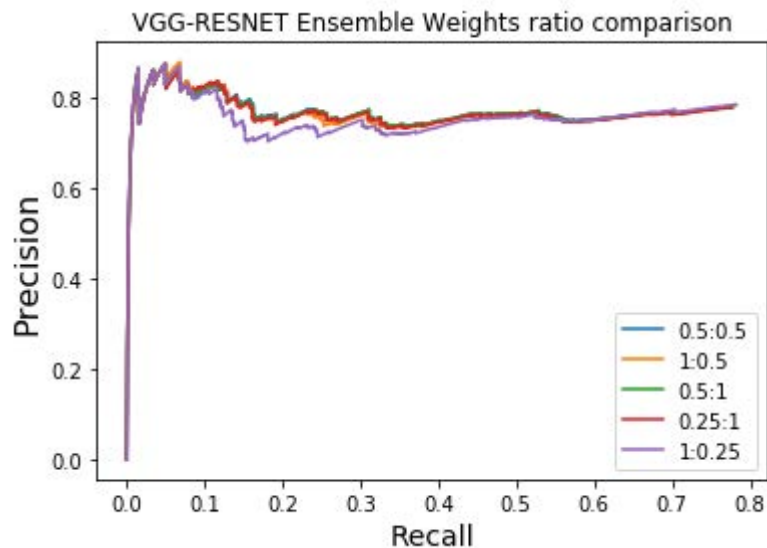
After creating the weighted feature representation, the top K recommendations were determined based on the top K scores and the (idx features) cosine-similarity between the test and training images.

The ensemble strategy seemed to perform a little bit better when performance of our ensemble model was compared to that of the solo VGG and Resnet models.



Ensemble model vs stand-alone VGG and Resnet comparison

We used this strategy for other weight ratios since we saw an improvement in accuracy, and the results are shown below.



Ensemble model precision vs recall comparison for different weights combinations

We tested a variety of weight combinations, but there was little improvement in accuracy, so we switched to a different approach.

## **CNN Classifier Based Retrieval technique (CCBR)**

We worked with a range of feature representations using pre-trained models, so we tested a different method from the previous ones, where we first categorise the input image and then offer recommendations based on the anticipated class.

## Why CCBR technique worked better than pre-trained models?

There may be two primary causes, including

Before generating feature vectors to forecast the class of the input image, the CNN model was employed as a classifier, which boosted the likelihood of receiving accurate recommendations. In contrast, pre-trained models computed similarity with all training photos independent of the class to which the images belonged and employed direct feature representations.

However, the CNN model was specifically trained on the fashion dataset, enabling it to learn the specialised feature representation of these photographs. The pre-trained models were taught on Image-Net, where they may have learned bigger and generic image feature representation.





Bag input image (left) and top 5 recommendations

## Methodology

At first, we attempted to frame the issue as a ranking task with lambdarank serving as the goal function. However, more testing revealed that treating it as a straightforward binary classification issue, predicting a product's clickout likelihood using the binary cross-entropy goal, and ranking the goods using the prediction probabilities produce far better outcomes. The feature engineering portion of our solution was one of the most crucial aspects because it required a lot of time and effort. Based on user, session, product, clickout, and other pertinent variables, we have developed about 80 features. Using the context-aware post processing method we used, our findings have been improved even further.

### 3.1 Preparation Categorical variable encoding



was the primary data pretreatment technique we needed to use for this issue. The provided dataset has many category columns, the majority of which have several different categories. Due to the sparsity of the data, we chose label encoding as the preferred method for encoding categorical variables over methods like one-hot encoding.

### 3.2 Feature Extraction

The most crucial component of our solution is this. A thoughtfully designed feature may be able to explain the importance of a product within a query quite successfully. We can broadly divide features into two categories: click-out features and non-click-out features.

### **TSNE Dimension reduction**

t-SNE is a technique for dimensional analysis or reduction that is a short form of T-distributed Stochastic Neighbor Embedding. As the name suggests it is a nonlinear dimensionality technique that can be utilized in a scenario where the data is very high dimensional. We can also say this is a technique for visualizing high dimensional data into lower-dimensional space. For the first time, this technique was introduced by Laurens van der Maatens and Geoffrey Hinton in 2008.

How does t-SNE work?

It is a method for dimensionality reduction or, as was already indicated, a method for viewing highly dimensional data. In order to create low dimensional embeddings, this method minimises the Kullback-Leibler divergence by turning high dimensional data points into joint probabilities. This approach employs a non-convex cost function, which implies that the outcome might vary depending on how it is applied.

The proper working of t-SNE can be understood using the following steps:

- Firstly the algorithm of this technique first calculates the joint probabilities between the data points that represent the similarity between points.
- After the calculation of joint probability, it assigns the similarity between the data points on the basis of the calculated joint probability.
- After assigning the similarity, t-SNE represents the data points on lower dimensions on the basis of probability distribution until the minimum Kullback-Leibler divergence.

Kullback-Leibler divergence can be considered as a statistical distance where it represents the calculation of how one probability distribution is different from the other one.

## **Using t-SNE in Python**

Now you will apply t-SNE on an open source dataset and try to visualize the results. In addition, you will also visualize the output of PCA on the same dataset to compare it with that of t-SNE.

The dataset you will be using is Fashion-MNIST dataset and can be found [here](#) (don't forget to check it out!). The Fashion-MNIST dataset is a 28x28 grayscale image of 70,000 fashion products from 10 categories, with 7,000 images per category. The training set has 60,000 images, and the test set has 10,000 images. Fashion-MNIST is a replacement for the original MNIST dataset for producing better results, the image dimensions, training and test splits are similar to the original MNIST dataset. Similar to MNIST the Fashion-MNIST also consists of 10 labels, but instead of handwritten digits, you have 10 different labels of fashion accessories like sandals, shirt, trousers, etc.

Each training and test example is assigned to one of the following labels:

- 0 T-shirt/top
- 1 Trouser
- 2 Pullover

- 3 Dress
- 4 Coat
- 5 Sandal
- 6 Shirt
- 7 Sneaker
- 8 Bag
- 9 Ankle boot

You will apply the algorithms on the training set data. Use the `load_rohit_dataset()` function you just defined and pass the first argument as the location of the data files and second argument `kind` as `'train'` to read the training set data.

You can check the dimensions of the data using `shape` attribute. You will notice the training set has 60000 sample points and 784 features.

The variable `y_train` contains the labels of every sample marked as integers from 0 to 9.

Next, you will import the necessary libraries you will be using throughout the tutorial. To maintain reproducibility, you will define a random state variable `RS` and set it to 123.

```
def load_rhiti_dataset(path, kind='train'):
    import os
    import gzip
    import numpy as np

    """Load MNIST data from 'path'"""
    labels_path = os.path.join(path,
                                '%s-labels-idx1-ubyte.gz'
                                % kind)
    images_path = os.path.join(path,
                                '%s-images-idx3-ubyte.gz'
                                % kind)

    with gzip.open(labels_path, 'rb') as lbpath:
        labels = np.frombuffer(lbpath.read(), dtype=np.uint8,
                                offset=8)

    with gzip.open(images_path, 'rb') as imgpath:
        images = np.frombuffer(imgpath.read(), dtype=np.uint8,
                                offset=16).reshape(len(labels), 784)

    return images, labels
```

✓ 0.7s Python

```
X_train, y_train = load_rhiti_dataset('Downloads/datasets/mnist/load_rhiti_dataset', kind='train')

import time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.path_effects as PathEffects
%matplotlib inline

import seaborn as sns
sns.set_style('darkgrid')
sns.set_palette('muted')
sns.set_context('notebook', font_scale=1.5,
                rc={'lines.linewidth': 2.5})
RS = 123
```

Python

In order to visualize the results of both the algorithms, you will create a `fashion_scatter()` function which takes two arguments: 1. `x`, which is a

2-D numpy array containing the output of the algorithm and 2. `colors`, which is 1-D numpy array containing the labels of the dataset. The function will render a scatter plot with as many unique colors as the number of classes in the variable `colors`.

To make sure you don't burden your machine concerning memory and time you will only use the first 20,000 samples to run the algorithms on.

```
# Utility function to visualize the outputs of PCA and t-SNE

def fashion_scatter(x, colors):
    # choose a color palette with seaborn.
    num_classes = len(np.unique(colors))
    palette = np.array(sns.color_palette("hls", num_classes))

    # create a scatter plot.
    f = plt.figure(figsize=(8, 8))
    ax = plt.subplot(aspect='equal')
    sc = ax.scatter(x[:,0], x[:,1], lw=0, s=40, c=palette[colors.astype(np.int)])
    plt.xlim(-25, 25)
    plt.ylim(-25, 25)
    ax.axis('off')
    ax.axis('tight')

    # add the labels for each digit corresponding to the label
    txts = []

    for i in range(num_classes):
        # Position of each label at median of data points.

        xtext, ytext = np.median(x[colors == i, :], axis=0)
        txt = ax.text(xtext, ytext, str(i), fontsize=24)
        txt.set_path_effects([
            PathEffects.Stroke(linewidth=5, foreground="w"),
            PathEffects.Normal()])
        txts.append(txt)

    return f, ax, sc, txts
```

Python

Also, don't forget to check whether the first 20000 samples cover samples from all the 10 classes.

We can now apply PCA to the data from the subset and display the results. With `n` components (Number of Principal Components to Keep) set to 4, you will utilise Sklearn's PCA. Although there are many options in sklearn that may be adjusted, you will utilise the default settings for the time being. Check out the PCA documentation on sklearn if you want to learn more about these parameters.

Notice the time taken by PCA to run on `x_subset` with 20000 samples.

Quite fast isn't it?

```
from sklearn.decomposition import PCA

time_start = time.time()

pca = PCA(n_components=4)
pca_result = pca.fit_transform(x_subset)

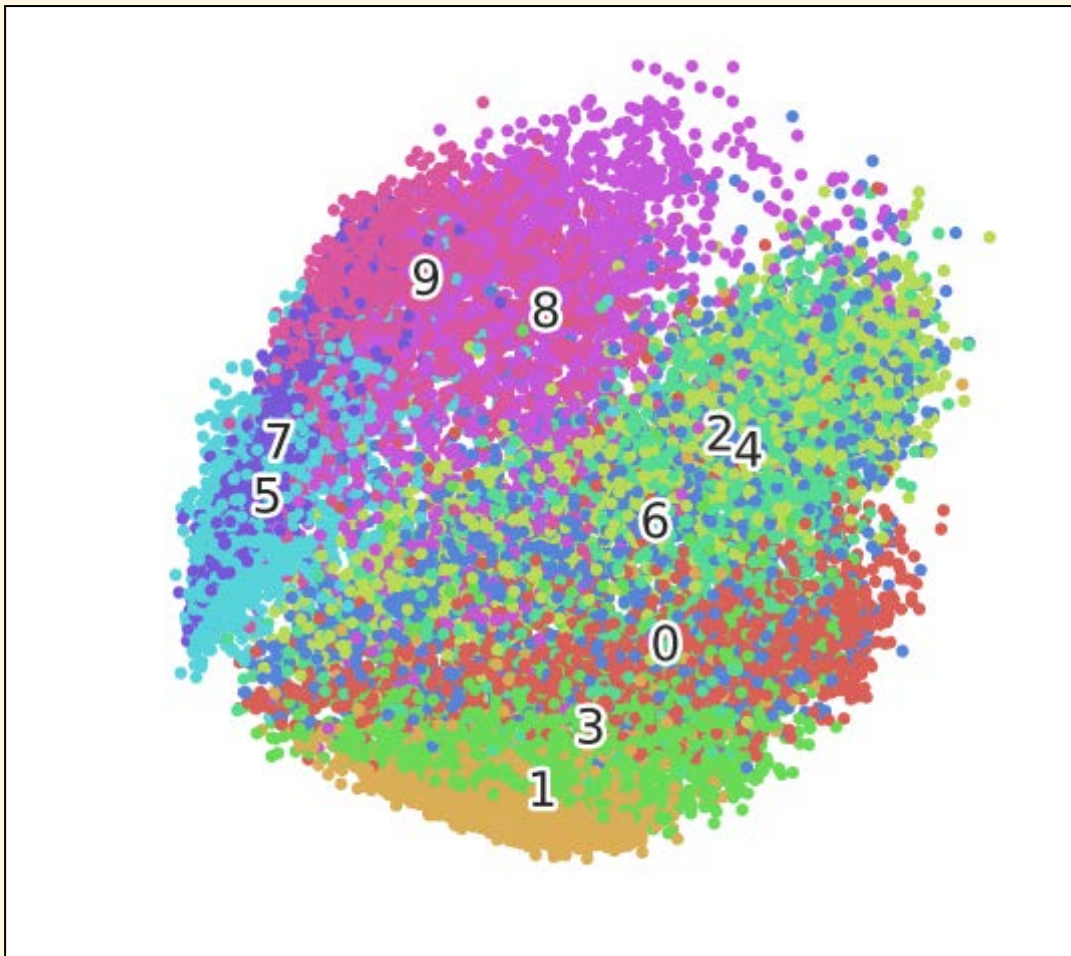
print ('PCA done! Time elapsed: {} seconds'.format(time.time()-time_start))
```



Now you will store all the four principal components in a new DataFrame `pca_df` and check the amount of variance of the data explained by these four components.

Note that the first and second principal components explain almost 48% of the variance in the data `x_subset`. You will use these two components for visualization by passing them in the `fashion_scatter()` function.





As you can see, PCA attempted to separate the various points and create clustered groups of points that were similar. The plot can also be used for preliminary analysis. Principal component algorithms (`pca1`, `pca2`, etc.) can use the principal components as features.

Now you will do the same exercise using the t-SNE algorithm.

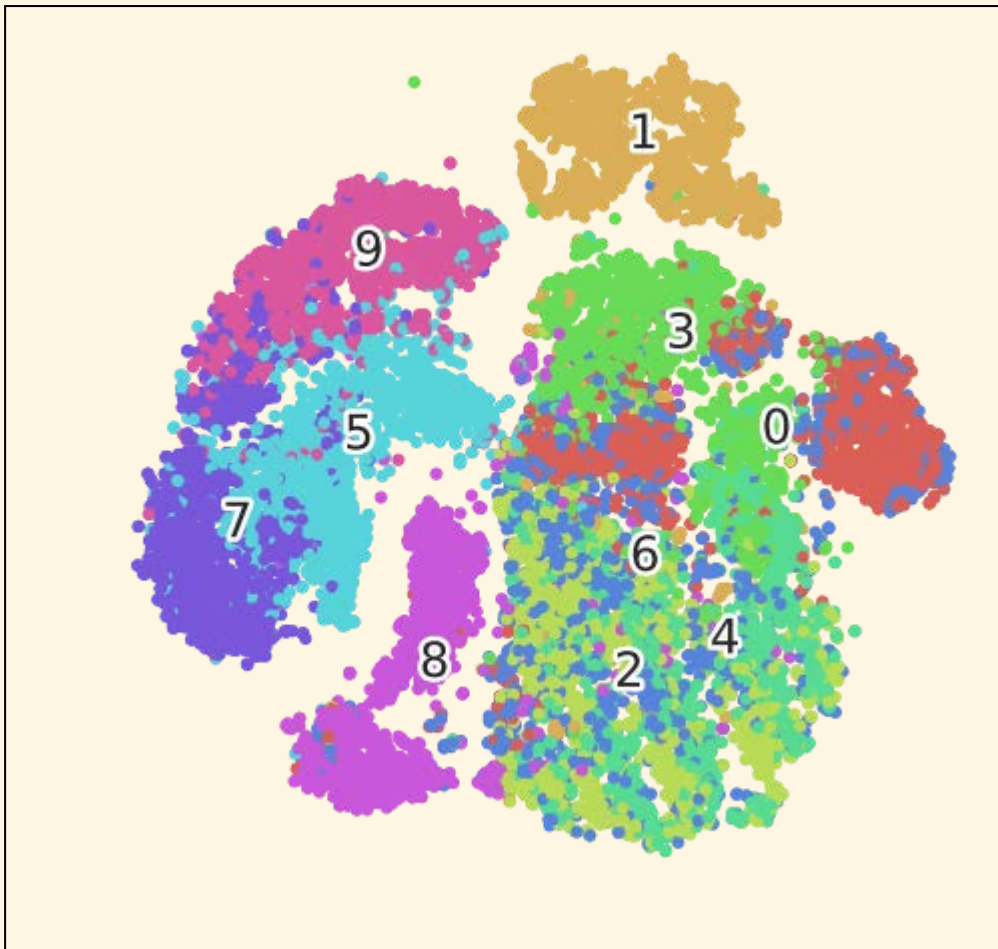
Scikit-learn has an implementation of t-SNE available, and you can check its documentation [here](#). It provides a wide variety of tuning parameters for t-SNE, and the most notable ones are:

- **n\_components** (default: 2): Dimension of the embedded space.
- **perplexity** (default: 30): The perplexity is related to the number of nearest neighbors that are used in other manifold learning algorithms.  
  
Consider selecting a value between 5 and 50.
- **early\_exaggeration** (default: 12.0): Controls how tight natural clusters in the original space are in the embedded space and how much space will be between them.
- **learning\_rate** (default: 200.0): The learning rate for t-SNE is usually in the range (10.0, 1000.0).
- **n\_iter** (default: 1000): Maximum number of iterations for the optimization. Should be at least 250.
- **method** (default: 'barnes\_hut'): Barnes-Hut approximation runs in  $O(N \log N)$  time. method='exact' will run on the slower, but exact, algorithm in  $O(N^2)$  time.

You will run t-SNE on `x_subset` with default parameters.

It can be seen that t-SNE takes considerably longer time to execute on the same sample size of data than PCA.

Visualizing the output of t-SNE using `fashion_scatter()` function:



Over the PCA visualization we created earlier, it is clearly visible that there is a significant improvement. The numerals are quite obviously grouped together into their own tiny group, as you can see.

```
pca_df = pd.DataFrame(columns = ['pca1', 'pca2', 'pca3', 'pca4'])

pca_df['pca1'] = pca_result[:,0]
pca_df['pca2'] = pca_result[:,1]
pca_df['pca3'] = pca_result[:,2]
pca_df['pca4'] = pca_result[:,3]

print ( 'Variance explained per principal component: {}'.format(pca.explained_variance_ratio_))
```

If you would now use a clustering algorithm to pick out the separate

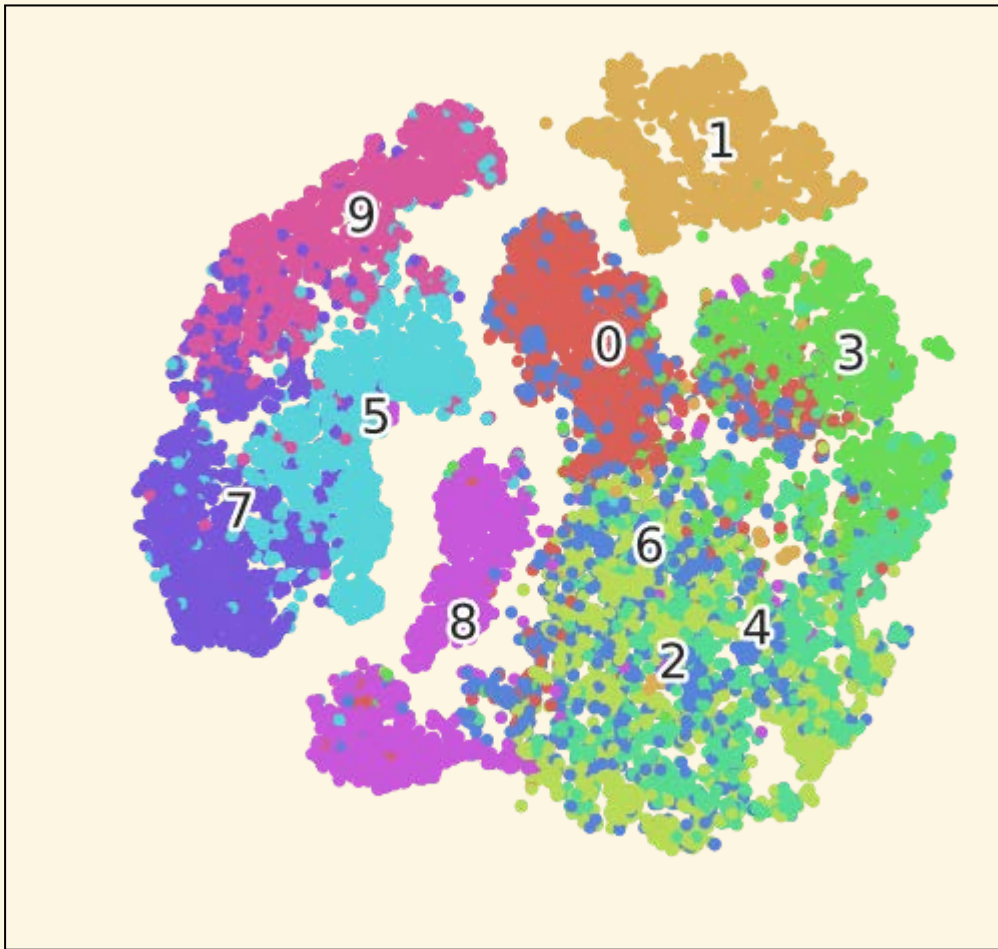
clusters, you could probably quite accurately assign new points to a label.

Scikit-learn's documentation of t-SNE explicitly states that:

*If the number of features is very high, it is strongly advised to use another dimensionality reduction method to reduce the number of dimensions to a manageable number (e.g., 50), such as PCA for dense data or TruncatedSVD for sparse data. By doing this, some noise will be reduced and the computation of pairwise distances between samples will be sped up.*

Now that you've heard this advice, you'll really heed it and minimise numerous values before submitting the data to the t-SNE algo, You'll once more utilise PCA for this. In order to perform the t-SNE, you must first cre-ate a new data-set with the fifty dimensions produced by the

PCA reduction algorithm.

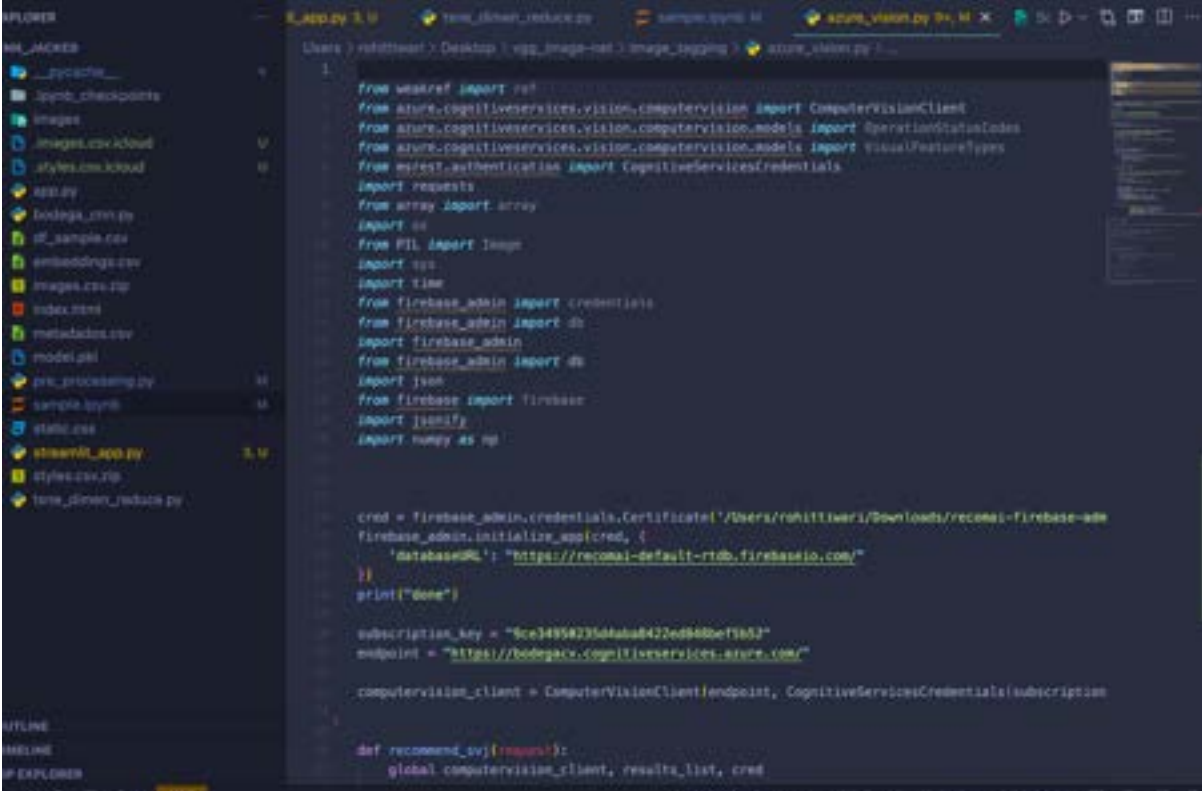


As we can clearly see that the plot is pretty much similar to the plot shown a few pages before this one but also there are certain minute changes, notice the data points are clustered more closely corresponding to label 0 (or images corresponding to T-shirt/top) as compared to the earlier plot. The duration of running t-SNE has also decreased.

## AZURE CUSTOM VISION FOR IMAGE TAGGING

One such service that we've recently been utilising is the Custom Vision Service, which is a component of the Cognitive Services offered by Microsoft and hosted by Azure.

Through the help of JSON-based REST calls over HTTP or, in many options, an alternative bigger-level client library, we can integrate the various specific AI or machine learning tasks offered by the Cognitive Services into web, mobile, and desktop applications. These tasks are provided as services via an API.



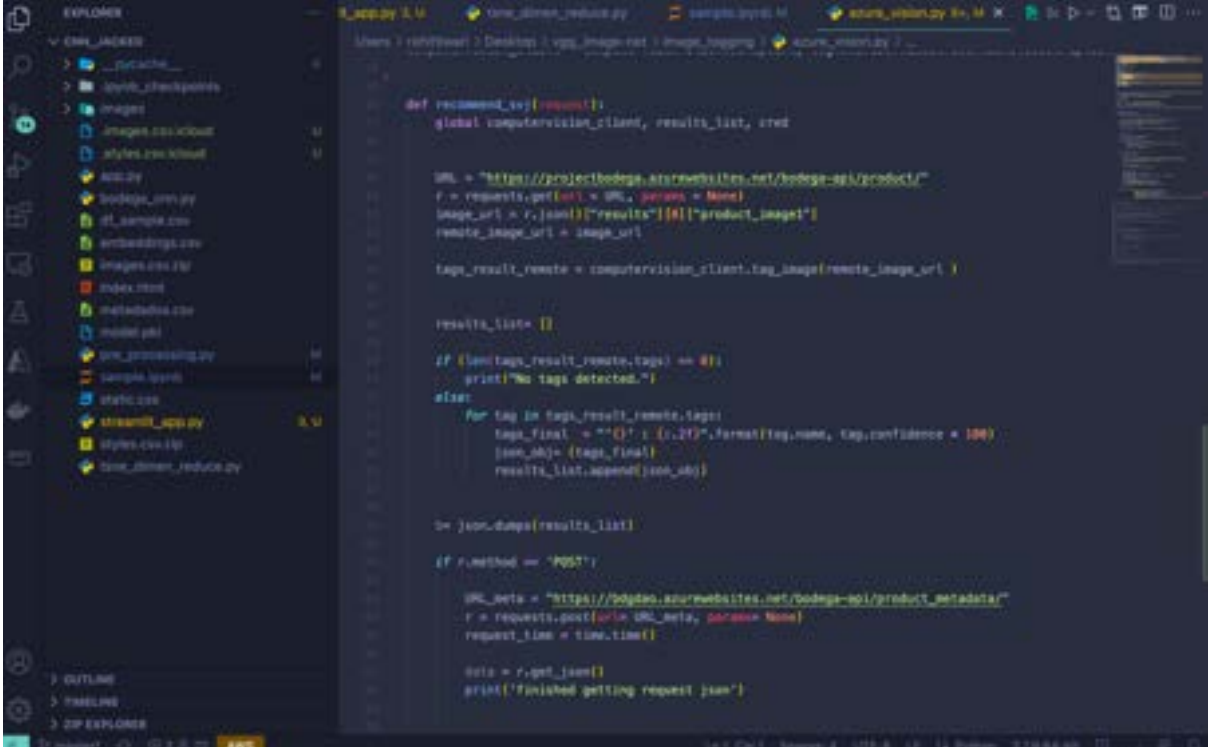
```
1
2 from urllib import request
3 from azure.cognitiveservices.vision.computervision import ComputerVisionClient
4 from azure.cognitiveservices.vision.computervision.models import OperationStatusCodes
5 from azure.cognitiveservices.vision.computervision.models import VisualFeatureTypes
6 from msrest.authentication import CognitiveServicesCredentials
7 import requests
8 from array import array
9 import os
10 from PIL import Image
11 import sys
12 import time
13 from firebase_admin import credentials
14 from firebase_admin import db
15 import firebase_admin
16 from firebase_admin import db
17 import json
18 from firebase import firebase
19 import jsonify
20 import numpy as np
21
22 cred = firebase_admin.credentials.Certificate('Users/rahitlweri/Downloads/recomai-firebase-adminkey-34954235d4aba8422ed948bef9b52.json')
23 firebase_admin.initialize_app(cred, {
24     'databaseURL': 'https://recomai-default-rtdb.firebaseio.com/'
25 })
26 print("done")
27
28 subscription_key = "34954235d4aba8422ed948bef9b52"
29 endpoint = "https://cognitive.azure.com/"
30
31 computervision_client = ComputerVisionClient(endpoint, CognitiveServicesCredentials(subscription_key))
32
33 def recommend_img(request):
34     global computervision_client, results_list, cred
```

The Payment for such services is dependent on the usage, it also provides various free tiers for experimentation purposes, development and low volume requirements. But basically the model is based on a pay-as-you-go model.

Computer Vision is basically the first of the two services which offer image recognition, it is more established and uses pre built model created by Microsoft. With the Custom Vision service, which is presently in preview, you may create and hone a model specific to a



specified picture domain.



```
def recommend_tag(request):
    global computer_vision_client, results_list, cred

    url = "https://projectibodega.azurewebsites.net/ibodega-api/product/"
    r = requests.get(url=url, params=None)
    image_url = r.json()["results"][0]["product_image"]
    remote_image_url = image_url

    tags_result_remote = computer_vision_client.tag_image(remote_image_url)

    results_list = []

    if (len(tags_result_remote.tags) == 0):
        print("No tags detected.")
    else:
        for tag in tags_result_remote.tags:
            tags_final = "{}".format(tag.name, tag.confidence * 100)
            json_obj = {"tags_final": tags_final}
            results_list.append(json_obj)

    json.dump(results_list)

    if r.method == "POST":
        url_meta = "https://ibodega.azurewebsites.net/ibodega-api/product_metadata/"
        r = requests.post(url=url_meta, params=None)
        request_line = time.time()

        data = r.get_json()
        print("Finished getting request json")
```

The benefit of a specialised model is that, although if there is more labour involved in finding photographs, labelling them, training, and improving the model, we will probably gain better accuracy when utilising only images from our selected subject. This is because the model is better able to distinguish between different images and stay focused by avoiding distractions from seemingly similar but actually quite diverse image themes.



## **Integrating the prediction API**

Up until this point, we've created and worked with our model using the Azure portal APIs. The actual and real purpose comes from the solution of the API, known as the prediction API, that enables to integrate it into our own apps. We may continue to utilise this strategy for the model's initial training and continuous refining.

But first, we must obtain three pieces of data from the portal, which we will then pass to the API in requests for identification and authorization.

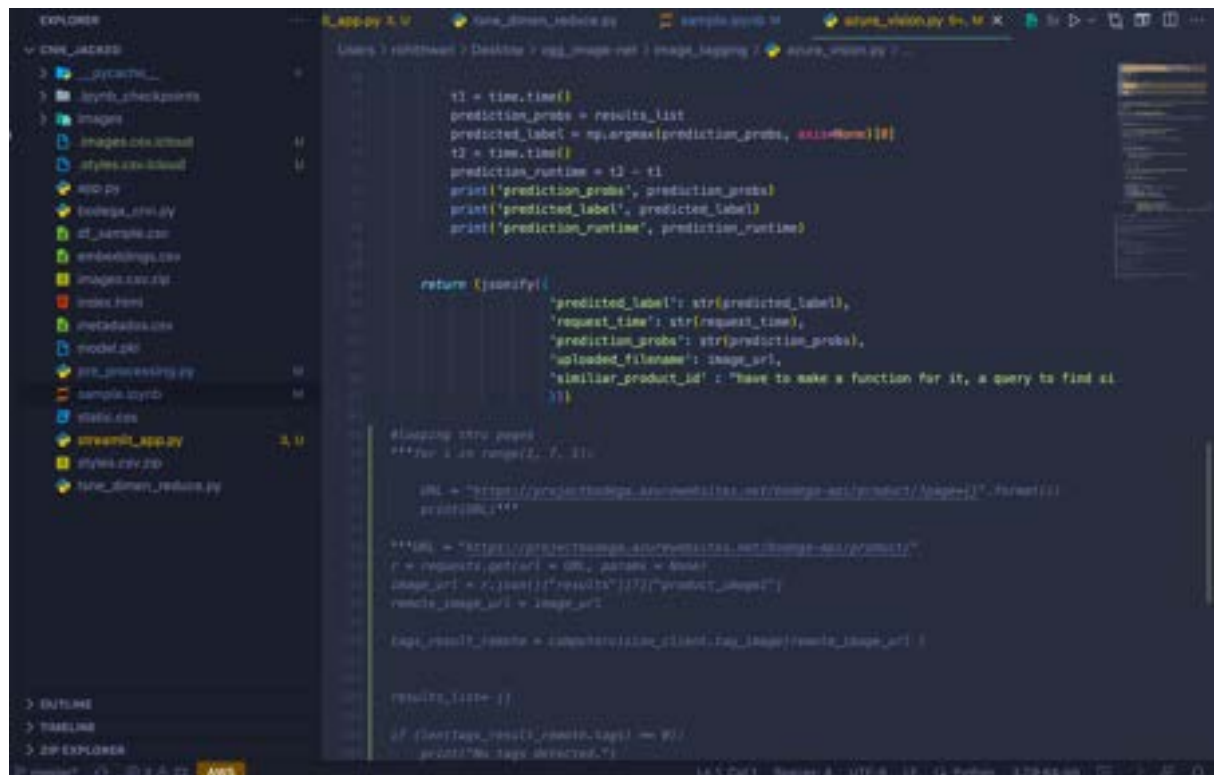
The project ID, project URL, and prediction API key are these.

The prediction API may be accessed in two different ways. The first method, and only if it is cross-platform, uses REST services that are called via HTTP, pass requests, and parse JSON-formatted responses.

On Windows, we also have the choice to rely on a client library, which

hides part of the HTTP request's inner workings and lets us operate at a higher, more strongly-typed level.

By detecting irrelevant picture types, such as quality control photos, Custom Vision has effectively assisted us in reducing mammography image quality concerns. This signifies a ground-breaking invention that will be used in millions of screenings worldwide.



## POST PROCESSING

We have used a context-aware post-processing approach to enhance the rating of products inside an impression. In essence, we need to select a cutoff value for product-context similarity. We rank the goods inside an impression using the model prediction probabilities based on condition that the product context similitude inside the impression is less than this cutoff. However, if the highest value exceeds the threshold, we

simply use the product-context similarity scores to rank the goods in the impression. Through the desired study that is the product context similaritycolumn inside the cutoff value, for a specific dataset is chosen. After evaluation, when the approval and required test labels are available, we have discovered that choosing a cutoff value in the range of 0.45 to 0.50 facilitates the best results.

## **HYPER PARAMETER TUNING USING Weights&Biases**

The W&B will take care of recording all the information of your models, from inputs (hyper-parameters) through outputs, by adding a few lines of code to your application (evaluation metrics). You would then have these logged facts presented to you in some really simple to understand graphs. All you have to do is relax with your favourite beverage as information is logged, then watch as your model is

constructed and those lovely metrics graphs produced automatically for you.

Additionally, so that you can always go back and study it later, you will have all the required (A-Z) information on the models or tests you conducted grouped in one location. These details are all accessible under one roof.

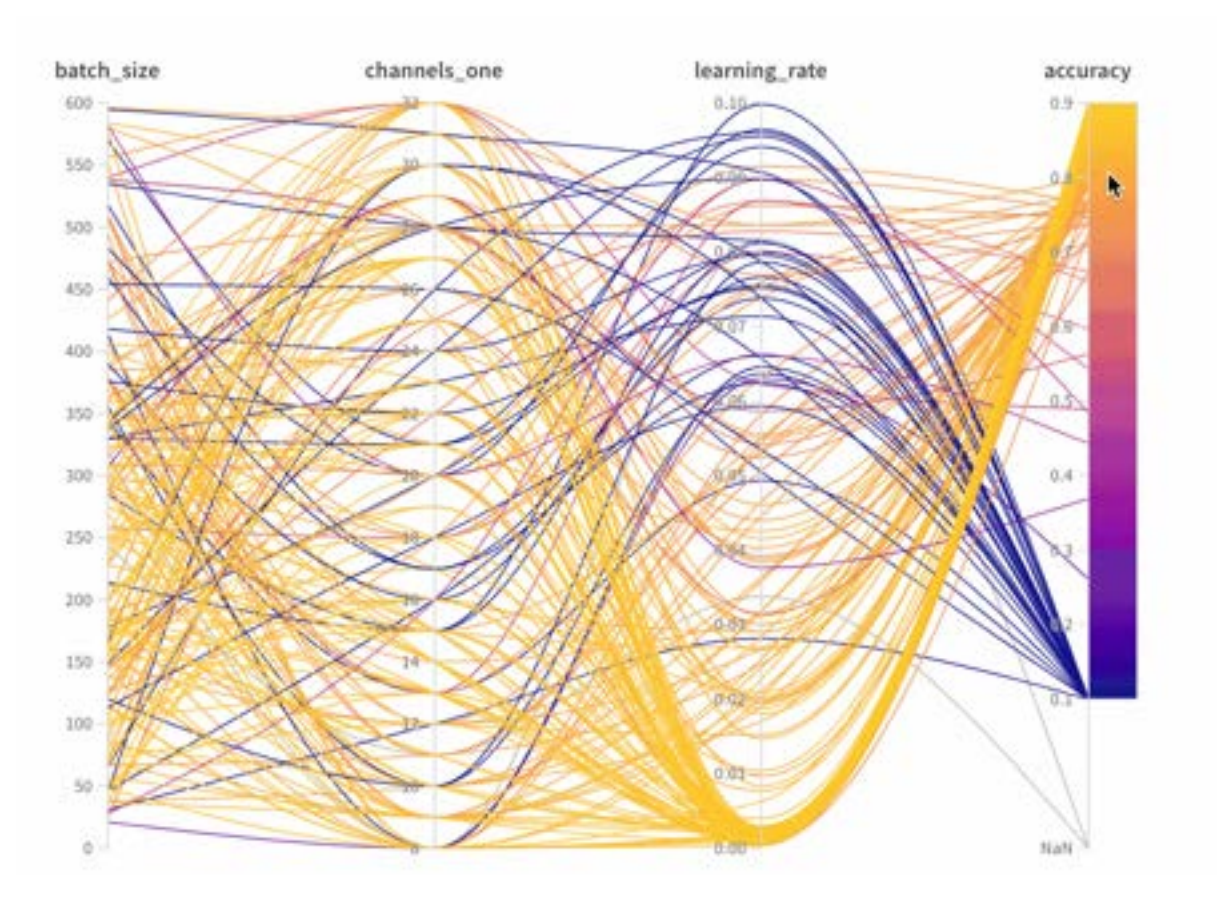
I think we can all agree that hyper-parameter tuning is a difficult process. When there are hundreds of hyper-parameters you may potentially adjust, it is just so challenging to observe the impact of tweaking the hyper-parameters on the optimum measure. Now, for hyper-parameter tuning, we are all familiar with Grid Search, Randomized Search, and Bayesian Optimization; however, we also know that all of these methods require a lot of CPU power. These methods don't function well with neural networks. I used to utilise my

own judgement to fine-tune the hyper-parameters while working really hard to identify changes in the optimising measure.

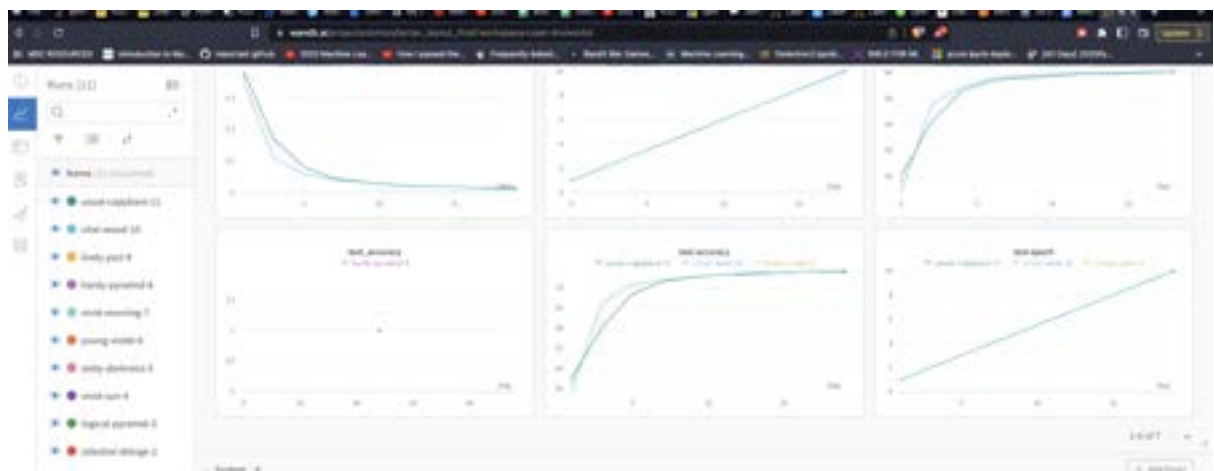
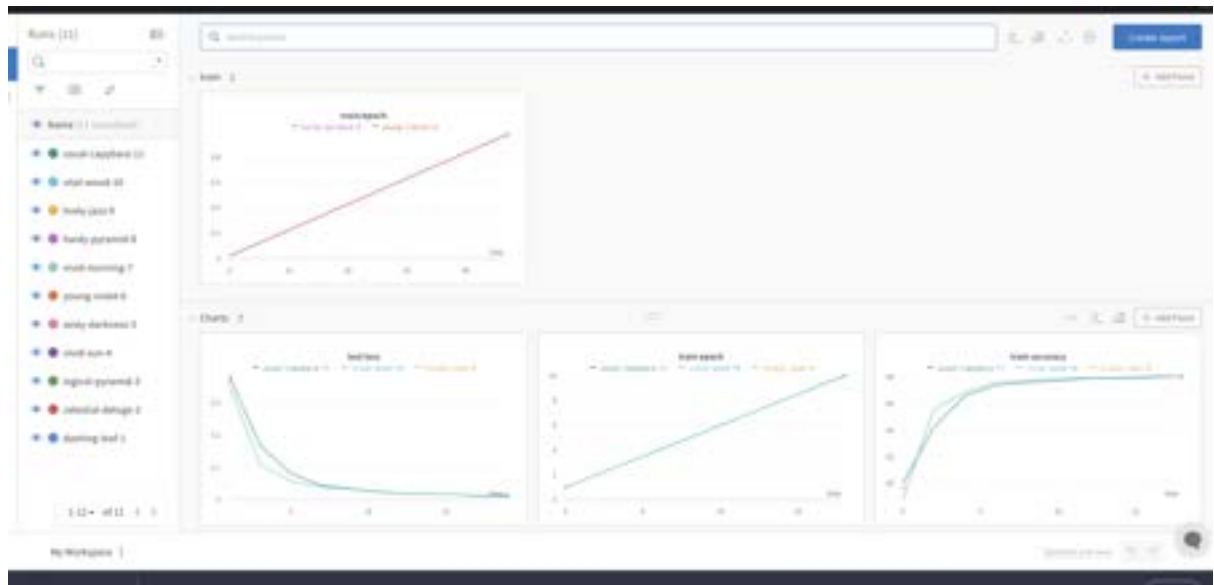
I think you will enjoy this cool feature if you are anything like us and want to keep an eye on how much CPU/GPU memory is being used while your model is running. The important system data, such CPU/GPU utilisation, temperature, and allocated memory, are automatically logged by W&B. Additionally, it generates attractive charts so you can see how your system is doing as your model is being trained. The following graphic shows how these metrics are shown on your dashboard.



Now, what can come to rescue is one nice feature of W&B — the hyper-parameter sweep. Within this so called hyper-parameter sweep, the W&B creates parallel coordinate plots, making it easier to spot relationship between the optimizing metric and a particular hyper-parameter. Here is the parallel-coordinate plot, here's how it looks like in the W&B dashboard.



**Here are the metrics for the model:**



*The term "hyper-parameter" in machine learning (ML) refers to a parameter that the model must be given before it can begin learning, which basically means the amount of units in a neural network that are hidden, the number of trees in a random forest, the K number of nearest neighbours in a KNN, etc.*



The point is that the HPs themselves need to be learned; for example, we don't know in advance whether a Neural Network with 10 hidden units or one with 1000 hidden units is better suited to solve our issue, or if a Random Forest with 50 trees or one with 100 trees would do better overall. Hence the requirement for HPT.

*While we give the model with HP, HP should not be confused with parameters, which are automatically learned inside the model's typical black-box.*

A brief search will reveal that the most important and common HPT strategies are:

1. Manually:
2. GS: Grid Search
3. Random Search
4. Bayesian optimization (BO).

One single fact—time—is the primary driver for such consideration for the HPT strategy. Finding the best HP for your issue can take a lot of

time, especially when you're working in a field where suggestions must be made instantly and updated with any new data you enter while working online.

We've created a reasonably basic example that demonstrates the difficulty of all this by contrasting two of these techniques.

- GS: a straightforward brute-force method.
- BO: a fully automated method that frequently outperforms other cutting-edge optimization algorithms.

WORKING:

```
from tensorflow.keras.applications import vgg16
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from keras.models import Model
from keras.applications.vgg16 import preprocess_input
from PIL import Image
import os
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

img_path = "/Users/rishittwari/Desktop/vgg_image-net/data/Apparel/Boys/Images/images_with_product_ids"
img_model_width, img_model_height = 224, 224
nb_closest_images = 10

vgg_model = vgg16.VGG16(weights='imagenet')

feat_extractor = Model(inputs=vgg_model.input, outputs=vgg_model.get_layer("fc2").output)
feat_extractor.summary()
```

```
Output exceeds the slide limit. Open the full output data in a text editor.
Model: "model"
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	5792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	347584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	598976
block3_conv3 (Conv2D)	(None, 56, 56, 256)	598976


```
store_images.py | model.py | vgg_image_new.py | video_recogn.py | streamlit_vgg.py |
vgg_image_new.py | vgg_model = vgg16.VGG16(weights='imagenet')
+ Code + Markdown | Run All | Clear Outputs of All Cells | Restart | Variables | Python 3.7.9 64-bit

files = [img_path + "r" + x for x in os.listdir(img_path) if ".jpg" in x]
print("number of images:", len(files))

number of images: 759

original = load_img(files[4], target_size=(img_model_width, img_model_height))
plt.imshow(original)
plt.show()
print("image loaded successfully!")

image loaded successfully!


```

```
store_images.py | model.py | vgg_image_new.py | video_recogn.py | streamlit_vgg.py |
vgg_image_new.py | vgg_model = vgg16.VGG16(weights='imagenet')
cosimilarities = cosine_similarity(imgs_features)
cos_similarities_df = pd.DataFrame(cosimilarities, columns=files, index=files)

(759, 759)

def retrieve_most_similar_products(given_img):
    print("original product:")
    original = load_img(given_img, target_size=(img_model_width, img_model_height))
    plt.imshow(original)
    plt.show()

    print("most similar products:")
    closest_imgs = cos_similarities_df[given_img].sort_values(ascending=False)[1:ns_closest_images+1].index
    closest_img_scores = cos_similarities_df[given_img].sort_values(ascending=False)[1:ns_closest_images+1]

    for i in range(0, len(closest_imgs)):
        original = load_img(closest_imgs[i], target_size=(img_model_width, img_model_height))
        plt.imshow(original)
        plt.show()
        print("similarity score: ", closest_img_scores[i])

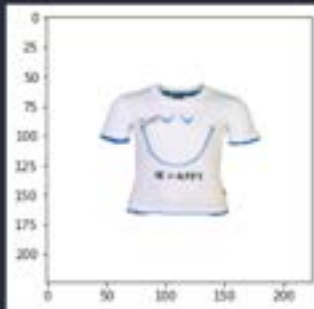
retrieve_most_similar_products(given_img)
```

```
retrieve_most_similar_products(files[45])
```

[12]

... original products:

✓



most similar products:

✓

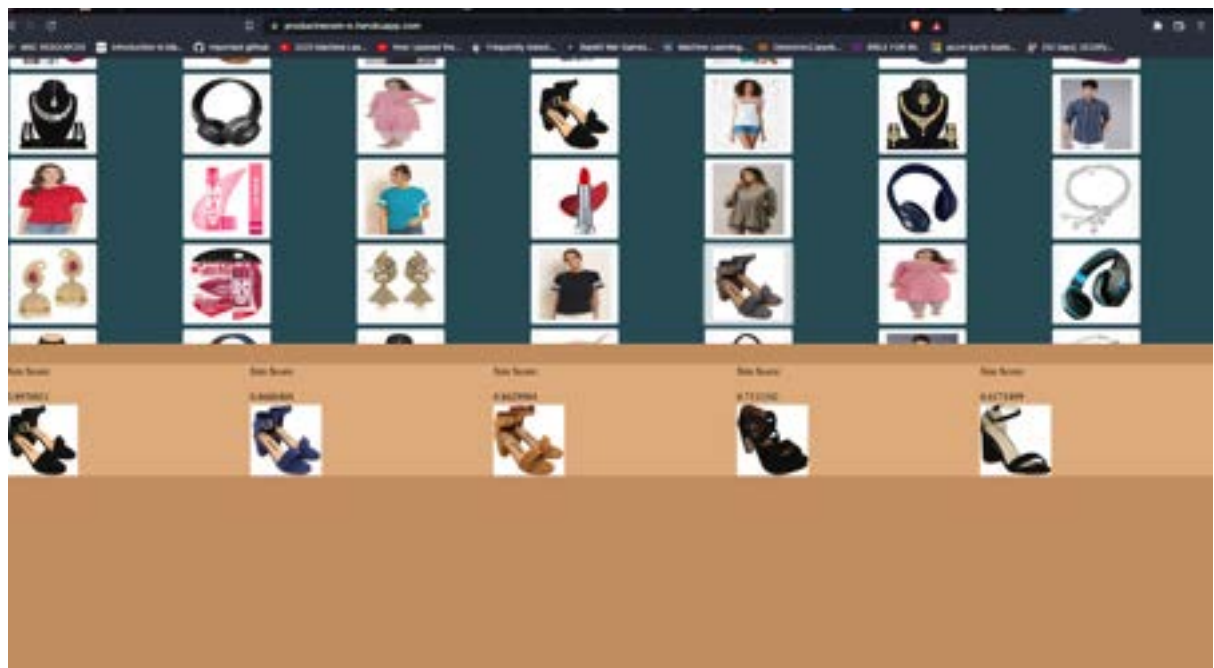
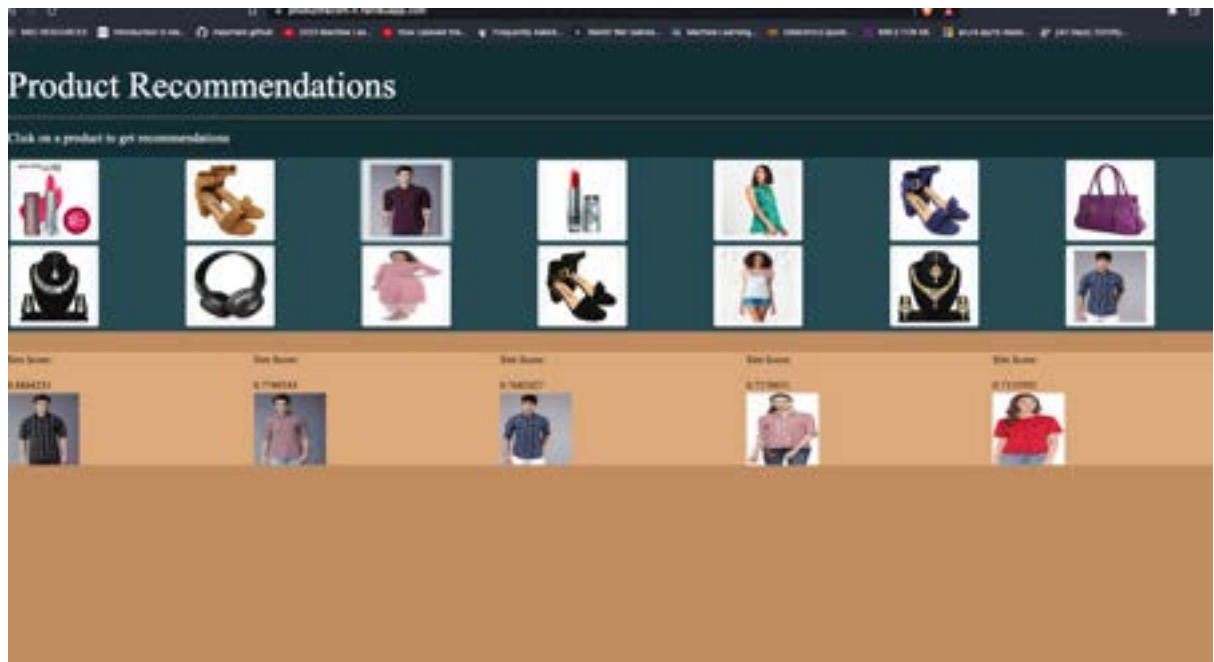


similarity score : 0.89567781

✓



## WEBAPP:



## CHAPTER-5: CONCLUSION AND FUTURE SCOPE

### RESULT

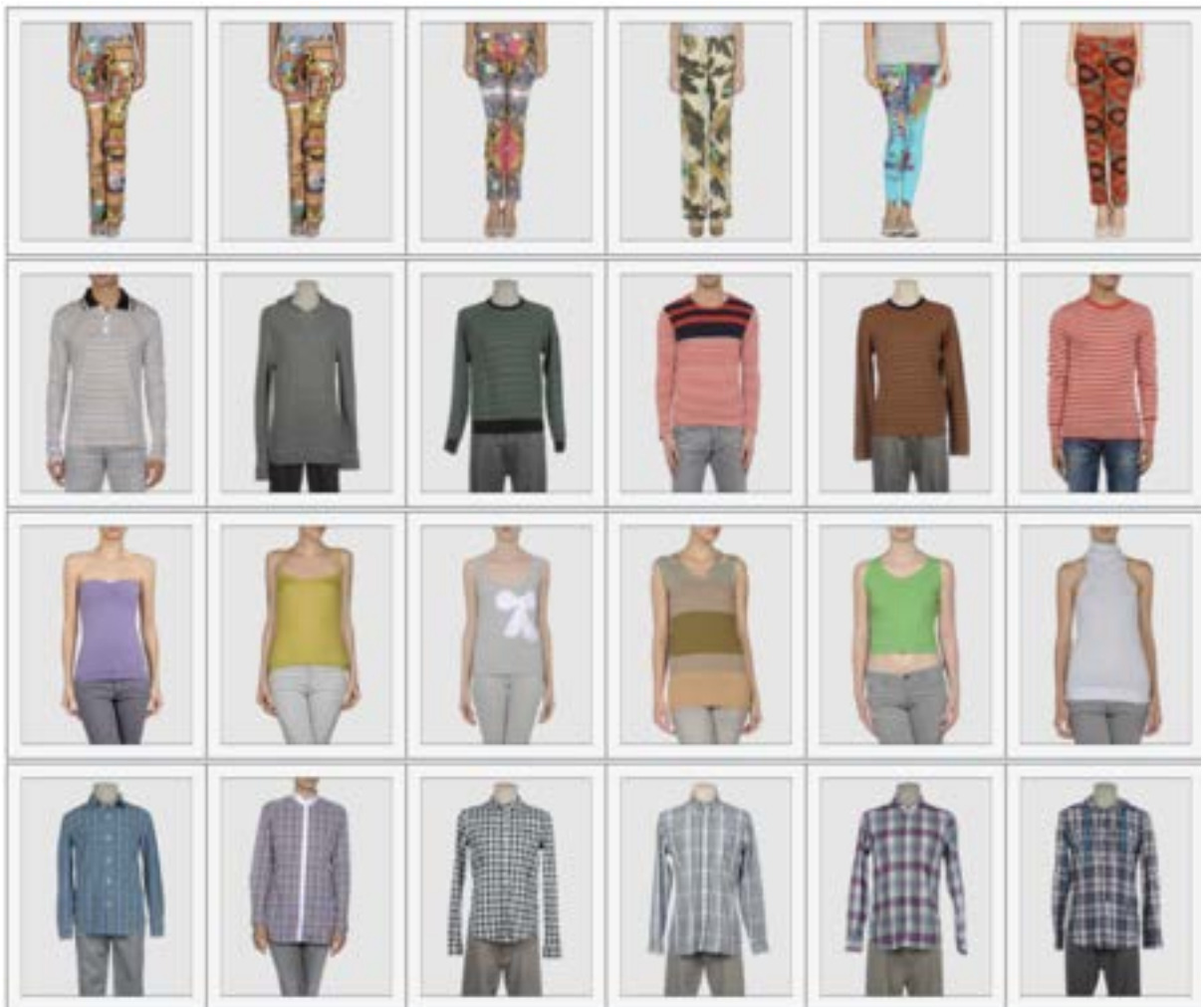
Using images, users of this online application may find products. If a user finds a product visually appealing then the user can view products similar to it through the help of the interface and can also submit the image to the model to view final products. The user will see products that the model found to be comparable to the provided image. Customers may choose to purchase the goods/products through the appropriate E-commerce websites if they so choose.

### CONCLUSION

One of the most premium ways to provide a good customer experience and suggestions is through the use of a product recommendation engine.

In this project, we presented an image-based attribute-aware recommender model for items' customised ranking with concurrent training of a ResNet50 component for adding image features into the recommender model. The model's performance significantly improved once the image attributes were included. Compared to all other image-based recommendation systems, ImgRecEtE performs better. In order to examine several methods of adding features to the model, including direct feature combination, adding a fine-tuning fully connected layer, and jointly training a portion of the picture

network, we also conducted an ablation research. A product recommendation engine aids in providing clients with the pertinent items they desire or need using machine learning, manual curation, and certain algorithms. It enables marketers to provide them real-time access to the generated recommendations. Product recommendations enhance the consumer experience by dynamically populating goods on websites, apps, contact centres, or emails as part of an e-commerce personalisation strategy.



Product recommendation systems can now through the help of specialised algorithms look after the largest commodity libraries. The engine can make



intelligent decisions about which procedures and screening methods to use in any given situation. The finest engines for improving user experiences for customers are those that provide product recommendations. A product recommendations engine may assist present clients with the proper goods they desire or need by utilising machine learning, manual curation, and particular algorithms. It enables marketers to provide clients relevant product suggestions right away. The customer experience is improved by dynamically adding products to websites, apps, contact centres, and emails as part of an e-commerce personalization strategy.

Thanks to specialised algorithms, product recommendation systems can now handle gargantuan commodity libraries. The capability of engine that is intelligently choosing which procedures and screening methods to use in every particular circumstance for every specific unique customer. The binary character of implicit input is emphasised by the probabilistic model that NCF learns.

We explored the use of generic matrix factorization, or GMF, to express and generalise MF under NCF. Through the use of a The user-item interaction function is learned using a multi-layer perceptron (MLP), NCF investigates the usage of DNNs for collaborative filtering. The last model we covered was the NeuMF neural matrix factorization model, which combines MF and MLP inside the NCF framework. It combines

the advantages of MF's linearity and MLP's non-linearity for modelling user-item latent structures.

FUTURE SCOPE:

### **Non Click-out Features**

These attributes are not affected by whether or not a specific product is selected. They are further classified into the following groups:

#### **Session-based Features:**

The session-specific aggregate and frequency characteristics are among these features. While frequency features include factors like product frequency, main colour frequency, etc. in a certain session, aggregate features include things like the mean, maximum, and lowest product pricing and start online date. For instance, a type of repeated clicks rate characteristic is the product session click percentage feature, which is the most significant feature of our top performing approach. It is the proportion of clicks on the product under consideration in a given session to clicks on all other items in that session.

#### **Query-based Features:**

Averaged mean, maximum, and lowest product prices as well as the launch date for a specific query. 4 One of the characteristics is Sourya and others.

#### **Global Static Features:**

This group of features includes details such overall user base who have interacted with a specific product, the amount of views the product has received, etc.

### **Popularity Features:**

These attributes, such as the popularity of the brand ID, the category ID, the primary color, etc., provide information about various aspects of the product.

### **User-tier features:**

This subcategory of features includes aggregate features that are specific to the User-tier, such as mean, maximum, and minimum product prices as well as start online date.

### **Ranking Features:**

The rating of a result inside impression of product price and date to start online is essentially what these attributes are.

Difference Characteristics: These features essentially represent how one attribute compares to another that is more aggregate in nature, such as the discrepancy for a given product's pricing and the mean commodity price of the session

### **REFERENCES:**

1. <https://towardsdatascience.com/likelihood-probability-and-the-math-you-should-know-9bf66db5241b>

2. [https://medium.com/@kaan\\_arik/unsupervised-learning-clustering-fa53032b640c](https://medium.com/@kaan_arik/unsupervised-learning-clustering-fa53032b640c)
3. <https://arxiv.org/abs/2207.12263>
4. <https://arxiv.org/abs/2101.06286>
5. <https://machinelearningmastery.com/image-augmentation-deep-learning-keras/>
6. <https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/>
7. <https://www.udemy.com/course/data-science-machine-learning-master-y/>
8. <https://www.packtpub.com/in/tech/machine-learning>
9. <https://nanonets.com/blog/information-extraction-graph-convolutional-networks/>