Santa Clara University
School of Engineering

# IMAGE BASED PRODUCT RECOMMENDATION SYSTEM

(PROJECT REPORT)

**Team Members**
Khetanshu chauhan
Uma Surakod
Vamshik Bellur Dayanand
Karthik Basavaraju
Sushruth Radhakrishna

## Table of Contents

# Introduction

## Objective

Most on-line shopping search engines are still largely depending on knowledge base and use key word matching as their search strategy to find the most likely product that consumers want to buy. This is inefficient in a way that the description of products can vary a lot from the seller's side to the buyer's side. we proposed an interactive product recommendation method, which considers not only the product diversity but also the visual similarity, to interactively capture a user's real intention and refine the product recommendation result based on the user's real product interests.

## What is the problem

Online shopping search engines use information from users' profiles (demographic filtering), similar neighbour's (collaborative filtering), and textual description (content-based model) to make recommendations, which easily generate irrelevant suggestions to users due to the ignorance of users' intentions and the visual similarity among products. In this paper, we present a smart search engine for on- line shopping. Basically, it uses images as its input, and tries to understand the information about products from these images. We first use a neural network to classify the input image as one of the product categories. Then use another neural network to model the similarity score between pair images, which will be used for selecting the closest product in our item database. We use Collaborative-filtering to calculate the similarity score for training data.

## Why this is a project related to this class

By using the image and finding similar pattern for required search/query in the online shopping search engine is the main idea of this project. Here we are searching for the similar item using data mining concept of collaborative filtering, content-based filtering.
Collaborative-filtering: Collaborative filtering, also referred to as social filtering, filters information by using the recommendations of other people. It is based on the idea that people who agreed in their evaluation of certain items in the past are likely to agree again in the future. A person who wants to see a movie for example, might ask for recommendations from friends. The recommendations of some friends who have similar interests are trusted more than recommendations from others. This information is used in the decision on which movie to see.
Content-based filtering: A content-based recommender works with data that the user provides, either explicitly (rating) or implicitly (clicking on a link). Based on that data, a user profile is generated, which is then used to make suggestions to the user. As the user provides more inputs or takes actions on the recommendations, the engine becomes more and more accurate.
Technique used for Content-based filtering is concepts of Term Frequency (***TF***) and Inverse Document Frequency (***IDF***) are used in information retrieval system and also content-based filtering mechanisms (such as a content-based recommender). They are used to determine the relative importance of a document / article / news item / movie etc.

## Why other approach is no good

Existing recommender systems use information from users' profiles (demographic filtering). Recommendation systems have been used tremendously academically and commercially, recommendations generated by these systems aim to offer relevant interesting items to users. Several approaches have been suggested for providing users with recommendations using their rating history, most of these approaches suffer from new user problem (cold-start) which is the initial lack of items ratings. Where the system can't recommend any product to a new user because of lack of user's history. The existing systems usually don't consider the sparsity problem when applying the Collaborative filtering. As a result, the time performance of Collaborative filtering recommendation is, therefore, constrained.

## Why you think your approach is better

In order to better guide the user to online shopping, increasingly importance has been attached to the research and application of e-commerce recommendation system. One of the most successful recommender technologies for e-commerce is collaborative filtering. Collaborative recommendation is typically based on the assumption that similar users should share similar taste in products. The system recommends products based on item ratings explicitly delivered by neighbours, where a new consumer is matched against the database to discover neighbours (i.e., existing customers with similar taste and preference). Due to its simplicity, collaborative filtering has been very successful in both research and practice.

Content-based filtering is using the technique to analyse a set of documents and descriptions of items previously rated by a user, and then build a profile or model of the user's interests based on the features of those rated items. Using the profile, the recommender system can filter out the suggestions that would fit for the user.
We are using advantage of both collaborative and content-based approach in combination to overcome the drawbacks. Also, we are availing the user to upload any image which contains the product. The product contained in the image might not be in the ecommerce product list. Hence other existing approaches fails to identify the related products of the given input image. Since we are using CNN to identify the product, Content Based Filtering can be used to identify the products which has similar attributes as that of the product in the given input image.

Area or scope of investigation
In our approach we are using limited data to train the neural network because of limited time we can extend it to large dataset for more flexibility. In future, we believe this opens up a promising line of work in using recommender systems for many fields.

# Theoretical bases and literature review

## Definition of the problem

Ecommerce website is one of the fast-growing market in the world. Everybody wants to use the internet as medium to order the products. It is important for the ecommerce websites to provide the best user experience to attract the users to shop in their websites. One of the major steps the ecommerce websites has taken to improve the user experience is to have a personalized recommendation system to each of its users.

The user shows an interest to a product by either clicking an image or link related to the product. If the user does not buy the product, in that session, then there is a high chance for the user to buy the product in his upcoming sessions. Instead of letting user scroll through pages to find the product, it is important for the website to provide this product in the very first page to have a better user experience. In-order to provide a better solution, the analysis of the image or link clicked is analysed. A proper analysis of the browsing history and finding out the link minded people would help the ecommerce website to provide a better user experience.

## Theoretical background of the problem

Ecommerce market is growing in a fast pace and various online strategies are employed to provide relevant advertisements and products to the user. It is important to provide those advertisements in which the user is interested than other. In-order to do so an analysis of each user using the website needs to be done. To address this problem, Researchers and Scientists have come up with a solution called Recommendation System to provide the users with the essential information.

The main goal of the recommendation system is to find the interests of the user and catering the user with his personal requirements. Recommendation System plays a major role in the ecommerce website. It helps to build a profile to the user based on the previous purchase history and browsing history and helps the website to better arrange the products in the website which helps the user to make easy purchase's and it gives better user experience.  Recommendation Systems are one of the most successful application of Data Mining and Machine Learning.

There are many different types of approaches developed to provide better Recommendation Systems. Following diagram shown below identifies the different approaches of the Recommendation Systems.

## K-means Clustering

*K*-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable *K*. The algorithm works iteratively to assign each data point to one of *K* groups based on the features that are provided. Data points are clustered based on feature similarity. The results of the *K*-means clustering algorithm are:

1. The centroids of the *K* clusters, which can be used to label new data
2. Labels for the training data (each data point is assigned to a single cluster)

Rather than defining groups before looking at the data, clustering allows you to find and analyze the groups that have formed organically. The "Choosing K" section below describes how the number of groups can be determined.
Each centroid of a cluster is a collection of feature values which define the resulting groups. Examining the centroid feature weights can be used to qualitatively interpret what kind of group each cluster represents.

This introduction to the $K$-means clustering algorithm covers:

- Common business cases where K-means is used
- The steps involved in running the algorithm
- A Python example using delivery fleet data

## Algorithm

The $K$-means clustering algorithm uses iterative refinement to produce a final result. The algorithm inputs are the number of clusters $K$ and the data set. The data set is a collection of features for each data point. The algorithms starts with initial estimates for the $K$ centroids, which can either be randomly generated or randomly selected from the data set. The algorithm then iterates between two steps:

### Data assignment step:

Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance. More formally, if $c_i$ is the collection of centroids in set $C$, then each data point $x$ is assigned to a cluster based on where $dist(\cdot)$ is the standard ($L_2$) Euclidean distance. Let the set of data point assignments for each $i^{th}$ cluster centroid be $S_i$.

### Centroid update step:

In this step, the centroids are recomputed. This is done by taking the mean of all data points assigned to that centroid's cluster.

The algorithm iterates between steps one and two until a stopping criteria is met (i.e., no data points change clusters, the sum of the distances is minimized, or some maximum number of iterations is reached).

This algorithm is guaranteed to converge to a result. The result may be a local optimum (i.e. not necessarily the best possible outcome), meaning that assessing more than one run of the algorithm with randomized starting centroids may give a better outcome.

## Choosing K

The algorithm described above finds the clusters and data set labels for a particular pre-chosen $K$. To find the number of clusters in the data, the user needs to run the $K$-means clustering algorithm for a range of $K$ values and compare the results. In general, there is no method for determining exact value of $K$, but an accurate estimate can be obtained using the following techniques. One of the metrics that is commonly used to compare results across different values of *K* is the mean distance between data points and their cluster centroid. Since increasing the number of clusters will always reduce the distance to data points, increasing *K* will *always* decrease this metric, to the extreme of reaching zero when *K* is the same as the number of data points. Thus, this metric cannot be used as the sole target. Instead, mean distance to the centroid as a function of *K* is plotted and the "elbow point," where the rate of decrease sharply shifts, can be used to roughly determine *K*.

A number of other techniques exist for validating *K*, including cross-validation, information criteria, the information theoretic jump method, the silhouette method, and the G-means algorithm. In addition, monitoring the distribution of data points across groups provides insight into how the algorithm is splitting the data for each *K*.

## Feature Engineering

Feature engineering is the process of using domain knowledge to choose which data metrics to input as features into a machine learning algorithm. Feature engineering plays a key role in *K*-means clustering; using meaningful features that capture the variability of the data is essential for the algorithm to find all of the naturally-occurring groups. Categorical data (i.e., category labels such as gender, country, and browser type) needs to be encoded or separated in a way that can still work with the algorithm.

Feature transformations, particularly to represent rates rather than measurements, can help to normalize the data. For example, in the delivery fleet example above, if total distance driven had been used rather than mean distance per day, then drivers would have been grouped by how long they had been driving for the company rather than rural vs. urban.

**Item Hierarchy:** In this approach, association of the product is identified and is recommended to the user. The diagram 3.1 shows that the user has bought the Printer from Best Buy in the past. Hence the user is now shown with the items which is bought frequently along with the Printer which in this case is the ink of the Printer.

**Attribute Based Recommendations:** In this approach, an analysis of the previous history is made and a matrix is generated out of it. The matrix contains all the attributes of the product. Now while recommending a new product, products with these attributes are chosen. In the diagram 3.1, the user likes the action movies starring Client Eastwood. There is a high chance that the user might like another action movie of Client Eastwood in this case “Good, Bad and the Ugly.”

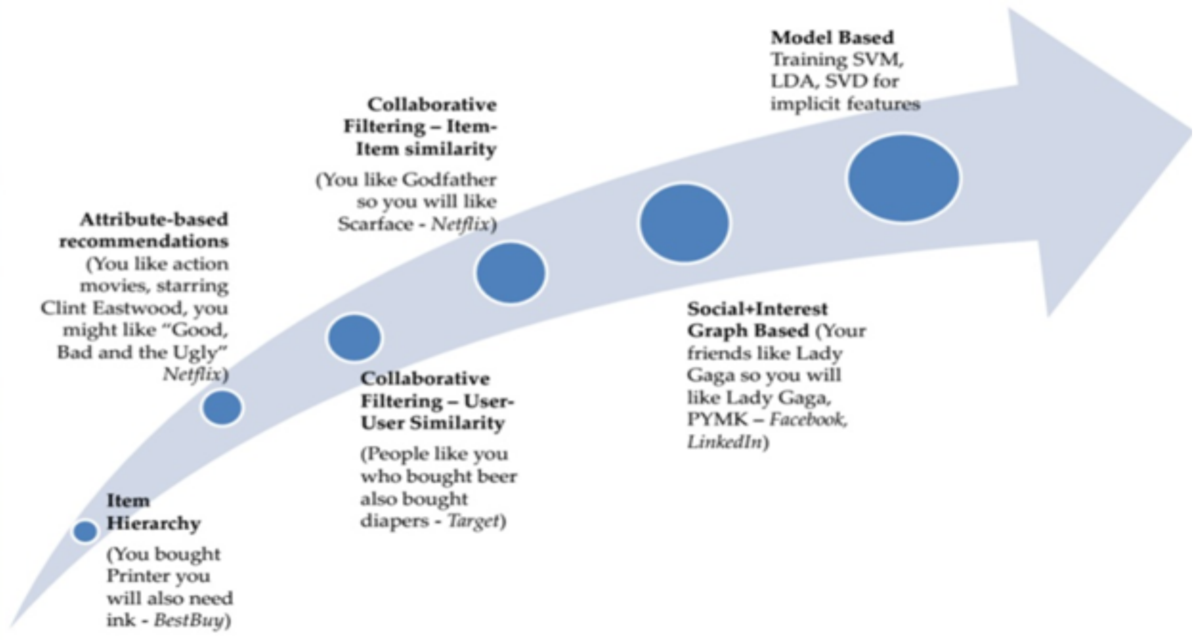

Model Based
Training SVM, LDA, SVD for implicit features
Collaborative Filtering – Item-Item similarity
(You like Godfather so you will like Scarface - Netflix)
Attribute-based recommendations
(You like action movies, starring Clint Eastwood, you might like "Good, Bad and the Ugly" Netflix)
Social+Interest Graph Based (Your friends like Lady Gaga so you will like Lady Gaga, PYMK – Facebook, LinkedIn)
Collaborative Filtering – User-User Similarity
(People like you who bought beer also bought diapers - Target)
Item Hierarchy
(You bought Printer you will also need ink - BestBuy)

**Figure showing the Different Approaches of Recommendation System**

**Collaborative Filtering Item-Item Similarity:** In this approach, products are recommended based on the similarity of the products. Products which are similar to the previous product purchased or liked, will be recommended to the user. There is no comparison with another user in the approach. In the diagram 3.1, the user has liked the Godfather which is a crime fiction movie. Hence the user will be recommended with another crime fiction movie, in this case it is Scarface.

**Collaborative Filtering User-User Similarity:** In this approach, products are recommended to the user based on the similar user likings. The system identifies another user who is similar and has same interests of the user and recommends the products which the user has not bought. Hence the comparison involves the likings of different users with same taste. In the diagram 3.2, the user has bought the beer and is now recommended with diapers because previously people bought beer and diaper together.

**Social + Interest Graph Based Approach**: In this approach, products are recommended to users based on the interests of the user's social network connection. People with same interests are connected in Social Media and hence might have same likings. Therfore a product bought by the user's close circle will be recommended to the user, it is not bought by the user.

**Model Based Approach:** In this approach, different Machine Learning models such as Support Vector Machine(SVM), Latent Dirichlet Allocation (LDA), Singular Value Decomposition will be used identify the essential features of the previous purchased products and a model will be constructed. This model will be used to recommend a new product to the user.

## Related research to solve the problem

Many Ecommerce websites like Amazon, Netflix use the personalized recommendation system to promote their product and to push the user to but the products such as books, movies, clothing and other consumer goods. Typical approaches followed in the recommendations are Content Based Filtering and Collaborative Filtering. The Content Based recommendation systems are used in the internet to have Web Personalizer. A mixture of both Content Based Filtering and also Collaborative Filtering is used to recommend a new product. The previous section discusses all the different approaches that is used to efficiently recommend the related products to the users.

[1] The paper uses the One Class Collaborative Filtering to recommend product in the online forum. The image features are extracted using the Deep Convolution Neural Network. Along with the extracted features from the image, user feedback ratings and product evolution trends are also considered to recommend products to the user. [2] The paper proposes a method to recommend handbags to each shopper. The paper uses the Joint learning of attribute Projection and One Class SVM classification based on the images of the shopper's clicked images. The features of the bag are extracted and are mapped to Projection Matrix. This projection matrix is used in conjunction with SVM Classifier to have a new way of recommendation system. [3] The paper proposes Image Based Recommendation System on Styles and Substitutes. A method to identify the visually similar looking products are recommended to the users in this paper. [4] Visual Similarity is used to recommend the products in an online shopping forum. Along with the product diversity, visual similarity is considered to capture the intention of the user and refine the product recommendation system.

[5] The paper discusses about the recommendation system used in the Amazon. Item-to-Item Collaborative Filtering is used to recommend the products in the Amazon where the focus is to find the similar items and not the customers. For each item purchased by the user and rated, the algorithm attempts to compute the similar items associated with the item and would recommend the same to the user. [6] The paper focuses on improving the recommendation performance by learning 'fashion aware' by training the image representations and the recommendation system jointly. The paper is similar to the work of Siamese CNNs and was able to show improvements compared to state of the art recommendation techniques such as BPR and the variants that make use of pre-trained visual features.

[7] The paper uses the convolution network to classify the input image to one of the product categories. It uses another convolution network to identify the similar product of the input image. Similarity is calculated using Jaccard Similarity. [8] The paper emphasizes the method to improve the performance of the product recommendation with Sparse Data. Simplified Similarity Measure (SSM) is proposed to handle the sparse data. It helps to reduce the problem of finding the "dislike" group and thus helps in fast processing of the data. [9] The paper proposes the Deep Learning based Large Scale Visual Recommendation. A unified Deep Convolution Neural Network architecture VisNet is proposed in-order to learn embeddings to capture the notion of visual similarity, across several semantic granularities.

## Advantage/disadvantage of those research

**Advantage**: Most of the approaches followed by the papers mentioned above have proved that their approach works better. Some of the papers proved that their work outperforms the state of the art approaches in uncovering rating dimensions and modelling user item interactions. All of the papers showed that given a large data set, the accuracy is more and hence more efficient the recommendation system is. With CNNs, it is able to extract the features of the image and using the suitable Machine Learning classifiers helped to achieve the more accurate recommended system.

**Disadvantage:** Most of the related works did not consider the problem of sparse data. The approaches consider would have lot of sparse data and did not mention how they would deal with such redundant sparse matrix. Recommendation Systems usually need a lot of data in-order to recommend appropriately. But they did not consider the problem of Cold Start.

## Your solution to solve this problem

Our approach is to provide a smart engine for online websites in-order to have better user experience. There are 2 different scenarios possible in our approach. In first scenario, a user can upload an image which has some product, then the system will identify the product and would then recommend the related products to the user. In the second scenario, user is shown the list of products. The list is arranged based on the user's previous history with the products browsed lately being shown in the top of the list.

In the first scenario, the user can upload any image which has some products. The image is then passed to a Convolution Neural Network (CNN) to scan the input image and to identify the distinct products in the image. The CNN then identifies the product and this product is passed as an input to the Recommendation System. The recommendation system now uses this product id and recommends the suitable products using the Content Based Filtering. The products which have the same content as that of the given input image is recommended. This is where our solution is different from other ecommerce websites recommendation system.

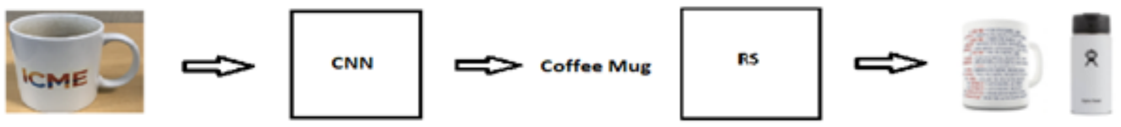


**Figure showing the Flow Diagram of Scenario 1 in Image Based Product Recommendation System.**

In the second scenario, the user is presented with a list of products based on the previous history. This is done using the identification of the positive data based on the previous history. An image of the product clicked by the user or added to wish-list shows that the user is interested in the product. These are the potential positive data for that user in the system. Once the positive data is identified, the recommendation system would use the collaborative filtering to better recommend the related products to the user. A mixture of Content based and Collaborative filtering will be used to have a better recommendation system.

## Where your solution different from others

In the existing ecommerce website, the user could not search based on uploading a new image which has the product. The user can search only the products displayed in the ecommerce website. Our approach allows the user to upload a new image. The CNN will then scan the input image uploaded by the user and then identifies the products in the image. This product info is passed to the recommendation system. The recommendation system uses the Content Based Filtering to provide the related recommendation systems.

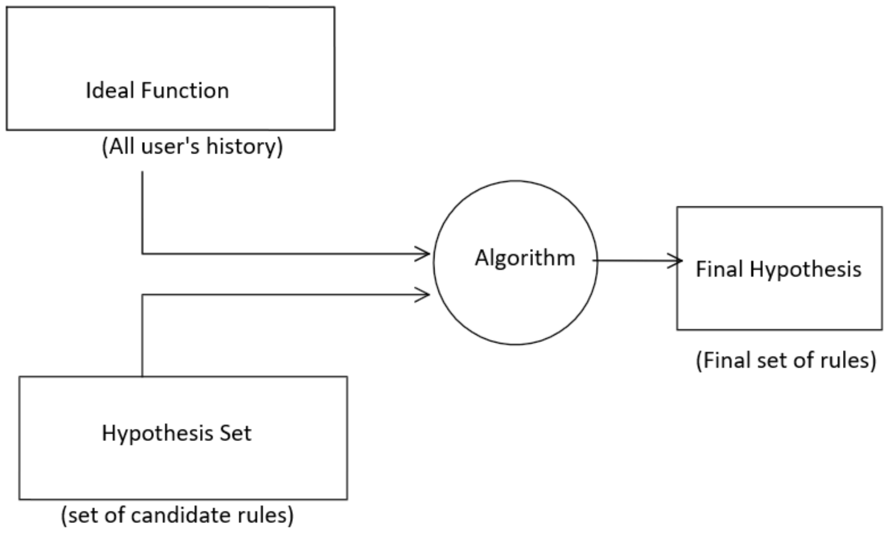
### Why your solution is better

In our approach, the user can upload any image which contains the product. The product contained in the image might not be in the ecommerce product list. Hence other existing approaches fails to identify the related products of the given input image. Since we are using CNN to identify the product, Content Based Filtering can be used to identify the products which has similar attributes as that of the product in the given input image.

## Hypothesis (or goals)

### Single/multiple hypothesis

The recommendation system should create final rules such that our function is as close as to the true function i.e. our model should give correct output to any unseen data. In our case, it would be a rule which would separate products the user may like from which he/she may not like. This can be achieved by finding correlation between all users' data and generating a final rule which would take new user's input and generates products which he/she may like.



### Positive or negative (only for proof correctness) hypothesis

Positive hypothesis would be all the set of rules which are formed based on the similar user's data. Negative hypothesis will be remaining rules. By using these rules, we can narrow down the products which he/she may like. Positive and Negative hypothesis will be used to shrinking the hypothesis space so that the model doesn't recommend product which is not relevant to the user. By this we can prove that the model's recommendation will be accurate all the time if system has sufficient users.

## Methodology

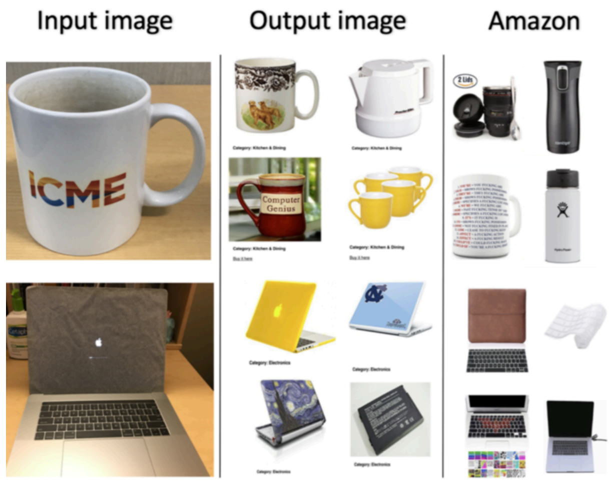
### How to generate/collect input data

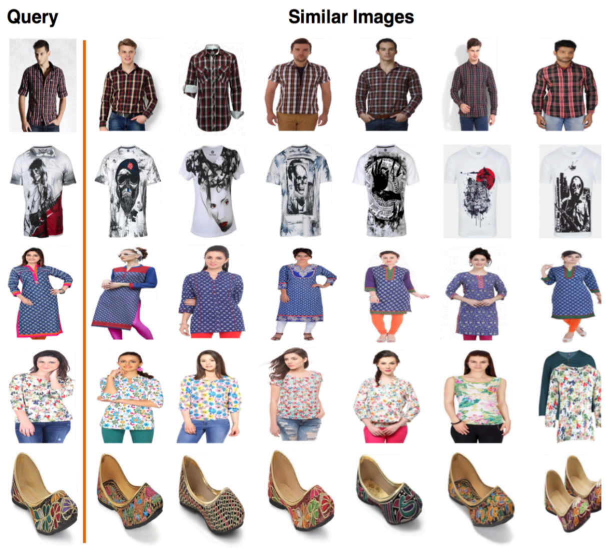
GENERATION OF DATA – WEB SCRAPING USING PYTHON

Selenium is used to send a request to the website from which the data is collected (images of the product). Python uses a Selenium driver to open the version of the web browser (chrome, Firefox, safari etc.)

### How to solve the problem

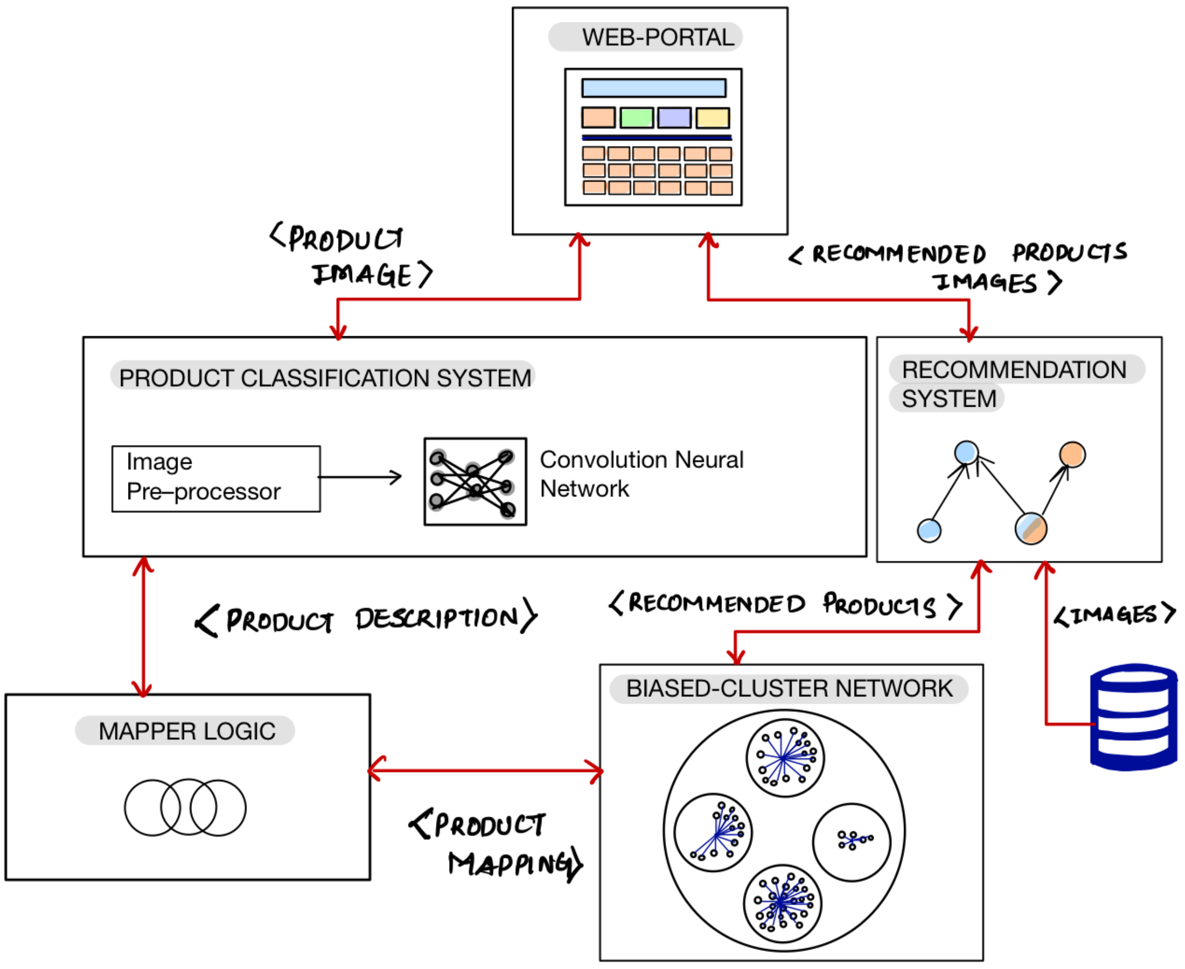
The problem could be solved by developing a smart search engine for online shopping which can use images as its input and tries to understand the information about products from these images. It should first use a neural network to classify the input image as one of the product categories. Then use another neural network to model the similarity score between pair images, which should be used for selecting the closest product in our e-item database. It should then use Jaccard similarity to calculate the similarity score for training data.



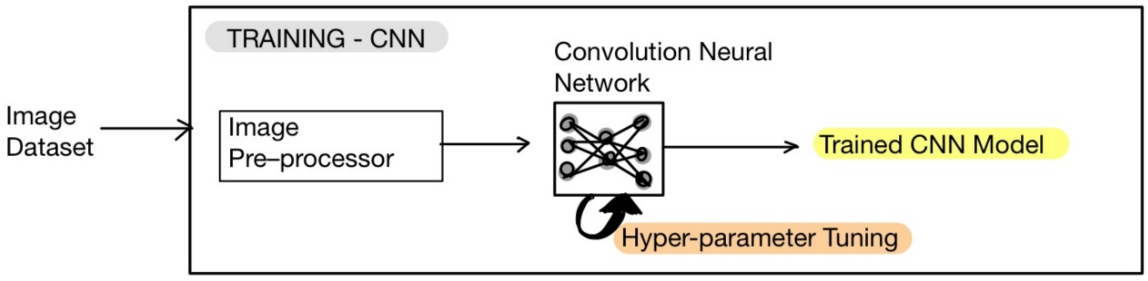


## Algorithm design

Following design represents the various modules of the project and how they are related to each other's through the interfaces,



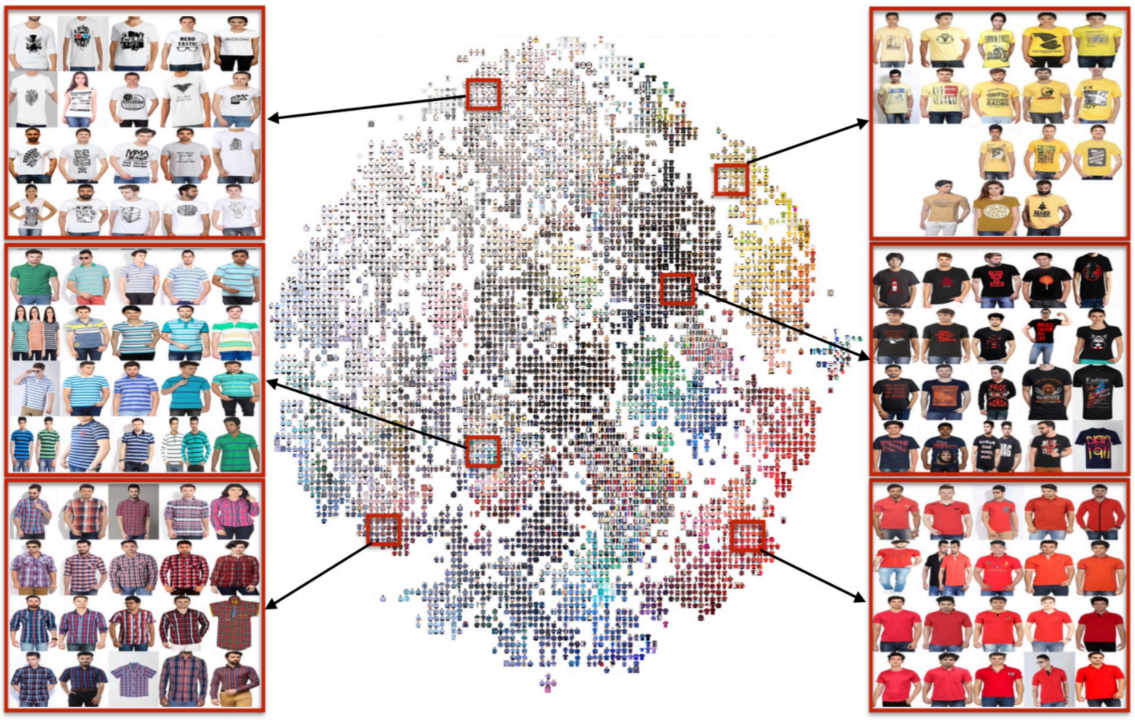
Foremost all, in the beginning, we need to train our convolutional neural network with the products dataset we have collected for the project. Data cleansing the other activities is also required while making the dataset ready for the training. Once the training is done the model can be used with the system to predict the product present in the images.



Web-portal is the main interface through which users can interact with the application. As an input, the user can select an available product on the application (web) and get the recommended product or they can upload a new image and get recommendations as in which are other product that is more likely related to the product present in the image.

If a product is selected amongst the available products, then the system would pass the product information to the "Recommendation system" and wait for the set of products with the images those are related to the product that was selected. Here, to find the similarity relationship to the "Recommendation system" would make use of the "Biased-Cluster-network" which would store these relationships (as depicted below).

The "Recommendation system" is a hybrid model of Collaborative filtering and Content based recommendation. The "Recommendation system" first finds the products which user may like based on other user's history (Collaborative filtering). Then these products are passed to second sub-model i.e. Content based recommendation. This model uses the output of Collaborative filtering as input and generates more recommendation which user may like. By incorporating both the techniques we can make the model more accurate. In addition to this we are using both Jaccard similarity and Pearson correlation coefficient to find similar users/products in the system. Since our model uses both models, it will give better recommendation accuracy when compared to other models. And while sending the information back to the "Web-portal" it would also pass the image files from the database of the recommended products through the FTP. There would be a web-service interface between the "Web-portal" and the "Recommendation system".



However, in the other scenarios where the user would upload a product image file, there the "Web-portal" would first pass the image to the "Product-Classification-system" to understand which products are present in the image. To process this "Product-Classification-system" would pre-process the image to the required format before passing it to the CNN (Convolutional Neural Network).

As a result, CNN would give a text output describing which all products are present in the image and share it with "Web-portal" and "Mapper-Logic" system. Now, if the products exist in the system, then there won't be any update made to the "Biased-Cluster-Network" and the databased. However, if the product is new, the "Mapper-Logic" would find its "Similarity-Score" with the existing "Biased-Cluster-Network" and save it in the related cluster for future recommendation. In parallel, for these kinds of scenarios after receiving the product details from the "Product-Classification-system" the "Web-portal" would ask for the related products from the "Recommendation system". Here, the "Recommendation system" would internally generate an "Approximate similarity score" and share the products as a recommendation which has a high probability of being related to the new product.

### Language used

- For **Convolutional Neural Network** development and training
  - Python 3
  - Libraries like
    - Keras 1.2.2+
    - TensorFlow R1.5
    - Pandas
    - NumPy
- For **Recommendation System**, **Mapper-Logic** and **Biased-Clustering Network** development
  - Python 3
  - Java 8
  - C++ 11
- For **Web-Portal** development
  - HTML
  - Bootstrap
  - Angular JS

## Pandas

### Why use Pandas?

Python has long been great for data munging and preparation, but less so for data analysis and modeling. *Pandas* helps fill this gap, enabling you to carry out your entire data analysis workflow in Python without having to switch to a more domain specific language like R.

Combined with the excellent IPython toolkit and other libraries, the environment for doing data analysis in Python excels in performance, productivity, and the ability to collaborate.

*Pandas* does not implement significant modeling functionality outside of linear and panel regression; for this, look to stats models and scikit-learn. More work is still needed to make Python a first class statistical modeling environment, but we are well on our way toward that goal.

## Library Highlights

- A fast and efficient DataFrame object for data manipulation with integrated indexing;
- Tools for reading and writing data between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format;
- Intelligent data alignment and integrated handling of missing data: gain automatic label-based alignment in computations and easily manipulate messy data into an orderly form;
- Flexible reshaping and pivoting of data sets;
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets;
- Columns can be inserted and deleted from data structures for size mutability;
- Aggregating or transforming data with a powerful group by engine allowing split-apply-combine operations on data sets;
- High performance merging and joining of data sets;
- Hierarchical axis indexing provides an intuitive way of working with high-dimensional data in a lower-dimensional data structure;
- Time series-functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data;
- Highly optimized for performance, with critical code paths written in Cythonor C.
- Python with *pandas* is in use in a wide variety of academic and commercial domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more.

## Numpy

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. If you are already familiar with MATLAB, you might find this tutorial useful to get started with Numpy.

## Arrays

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the *rank* of the array; the *shape* of an array is a tuple of integers giving the size of the array along each dimension.

## Array indexing

Numpy offers several ways to index into arrays.

- Slicing
    - Similar to Python lists, numpy arrays can be sliced.
- Integer array indexing
    - When you index into numpy arrays using slicing, the resulting array view will always be a subarray of the original array. In contrast, integer array indexing allows you to construct arbitrary arrays using the data from another array.
- Boolean array indexing

- Boolean array indexing lets you pick out arbitrary elements of an array. Frequently this type of indexing is used to select the elements of an array that satisfy some condition.

### Datatypes

Every numpy array is a grid of elements of the same type. Numpy provides a large set of numeric datatypes that you can use to construct arrays. Numpy tries to guess a datatype when you create an array, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype. Here is an example:

```
import numpy as np

x = np.array([1, 2])   # Let numpy choose the datatype
print(x.dtype)         # Prints "int64"

x = np.array([1.0, 2.0])   # Let numpy choose the datatype
print(x.dtype)             # Prints "float64"

x = np.array([1, 2], dtype=np.int64)   # Force a particular datatype
print(x.dtype)                         # Prints "int64"
```

### Array math

Basic mathematical functions operate elementwise on arrays, and are available both as operator overloads and as functions in the numpy module.

Apart from computing mathematical functions using arrays, we frequently need to reshape or otherwise manipulate data in arrays. The simplest example of this type of operation is transposing a matrix; to transpose a matrix, simply use the T attribute of an array object.

### Broadcasting

Broadcasting is a powerful mechanism that allows numpy to work with arrays of different shapes when performing arithmetic operations. Frequently we have a smaller array and a larger array, and we want to use the smaller array multiple times to perform some operation on the larger array.
Broadcasting two arrays together follows these rules:

1. If the arrays do not have the same rank, prepend the shape of the lower rank array with 1s until both shapes have the same length.
2. The two arrays are said to be *compatible* in a dimension if they have the same size in the dimension, or if one of the arrays has size 1 in that dimension.
3. The arrays can be broadcast together if they are compatible in all dimensions.
4. After broadcasting, each array behaves as if it had shape equal to the elementwise maximum of shapes of the two input arrays.
5. In any dimension where one array had size 1 and the other array had size greater than 1, the first array behaves as if it were copied along that dimension

Functions that support broadcasting are known as *universal functions*. Broadcasting typically makes your code more concise and faster.

## OpenCV-Python

Python is a general purpose programming language started by **Guido van Rossum**, which became very popular in short time mainly because of its simplicity and code readability. It enables the programmer to express his ideas in fewer lines of code without reducing any readability.

Compared to other languages like C/C++, Python is slower. But another important feature of Python is that it can be easily extended with C/C++. This feature helps us to write computationally intensive codes in C/C++ and create a Python wrapper for it so that we can use these wrappers as Python modules. This gives us two advantages: first, our code is as fast as original C/C++ code (since it is the actual C++ code working in background) and second, it is very easy to code in Python. This is how OpenCV-Python works, it is a Python wrapper around original C++ implementation.

And the support of Numpy makes the task easier. **Numpy** is a highly optimized library for numerical operations. It gives a MATLAB-style syntax. All the OpenCV array structures are converted to-and-from Numpy arrays. So whatever operations you can do in Numpy, you can combine it with OpenCV, which increases number of weapons in your arsenal. Besides that, several other libraries like SciPy, Matplotlib which supports Numpy can be used with this.
So OpenCV-Python is an appropriate tool for fast prototyping of computer vision problems.

## Machine learning: the problem setting

In general, a learning problem considers a set of n samples of data and then tries to predict properties of unknown data. If each sample is more than a single number and, for instance, a multi-dimensional entry (aka multivariate data), it is said to have several attributes or features.

We can separate learning problems in a few large categories:

- Supervised learning, in which the data comes with additional attributes that we want to predict (Click here to go to the scikit-learn supervised learning page).This problem can be either:
  - Classification: samples belong to two or more classes and we want to learn from already labelled data how to predict the class of unlabelled data. An example of classification problem would be the handwritten digit recognition example, in which the aim is to assign each input vector to one of a finite number of discrete categories. Another way to think of classification is as a discrete (as opposed to continuous) form of supervised learning where one has a limited number of categories and for each of the n samples provided, one is to try to label them with the correct category or class.
  - Regression: if the desired output consists of one or more continuous variables, then the task is called *regression*. An example of a regression problem would be the prediction of the length of a salmon as a function of its age and weight.

- Unsupervised learning, in which the training data consists of a set of input vectors x without any corresponding target values. The goal in such problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine the distribution of data within the input space, known as density estimation, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of *visualization* (Click here to go to the Scikit-Learn unsupervised learning page).

## Tools used

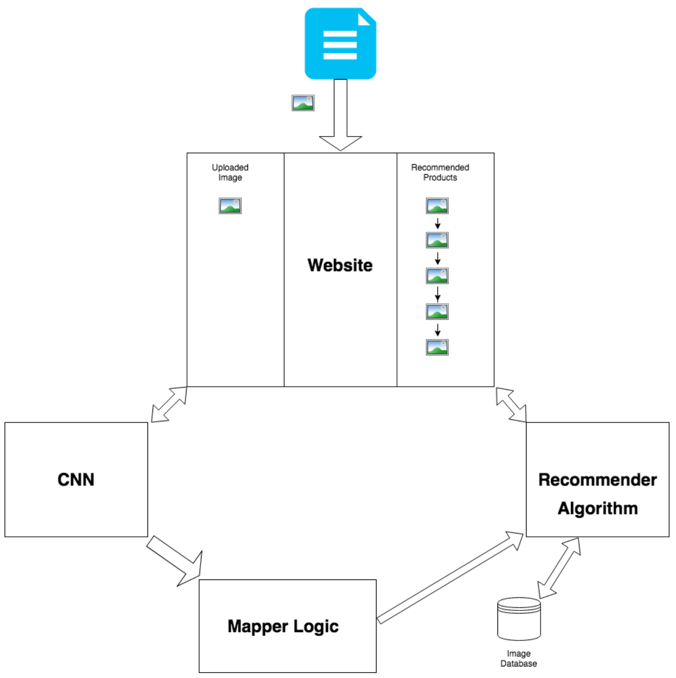
- Intelli-J
- Visual Studio
- Sublime Text
- Vi-Editor

## How to Get/Generate Data

Selenium module is used to send a request to the website from which the data is collected (images of the product). In Python, Selenium uses a driver to open the version of the web browser (chrome, Firefox, safari etc.) that can be controlled by python itself. This sends a request to a website and returns the response filled with HTML code which can be sorted through to find what is needed. The images are downloaded from the Google images which is set as the preferred website. By providing the upper limit, we download as many images from the site. Usually training a particular product requires a huge number of images of that particular product. We are having approximately 3000 images for each product for this project.

## How to generate output

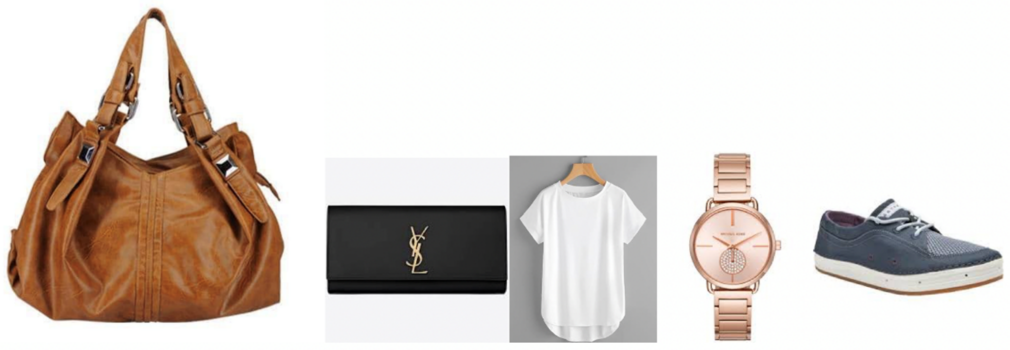
User uploads the image of the product/similar to the product on the web portal. The website gives him the exact match of the product he is looking for as well as the products related to the one which he has uploaded. The user gets a number of similar products that he is looking for which similar matched in design and styles that the system recommends him. The system also recommends the other products that are closely related to the product, for e.g., the user uploads a picture of the t-shirt that has a brand logo or a team logo on it, the system recommends similar t-shirts from the same brand/team along with shorts and shoes that goes with it.



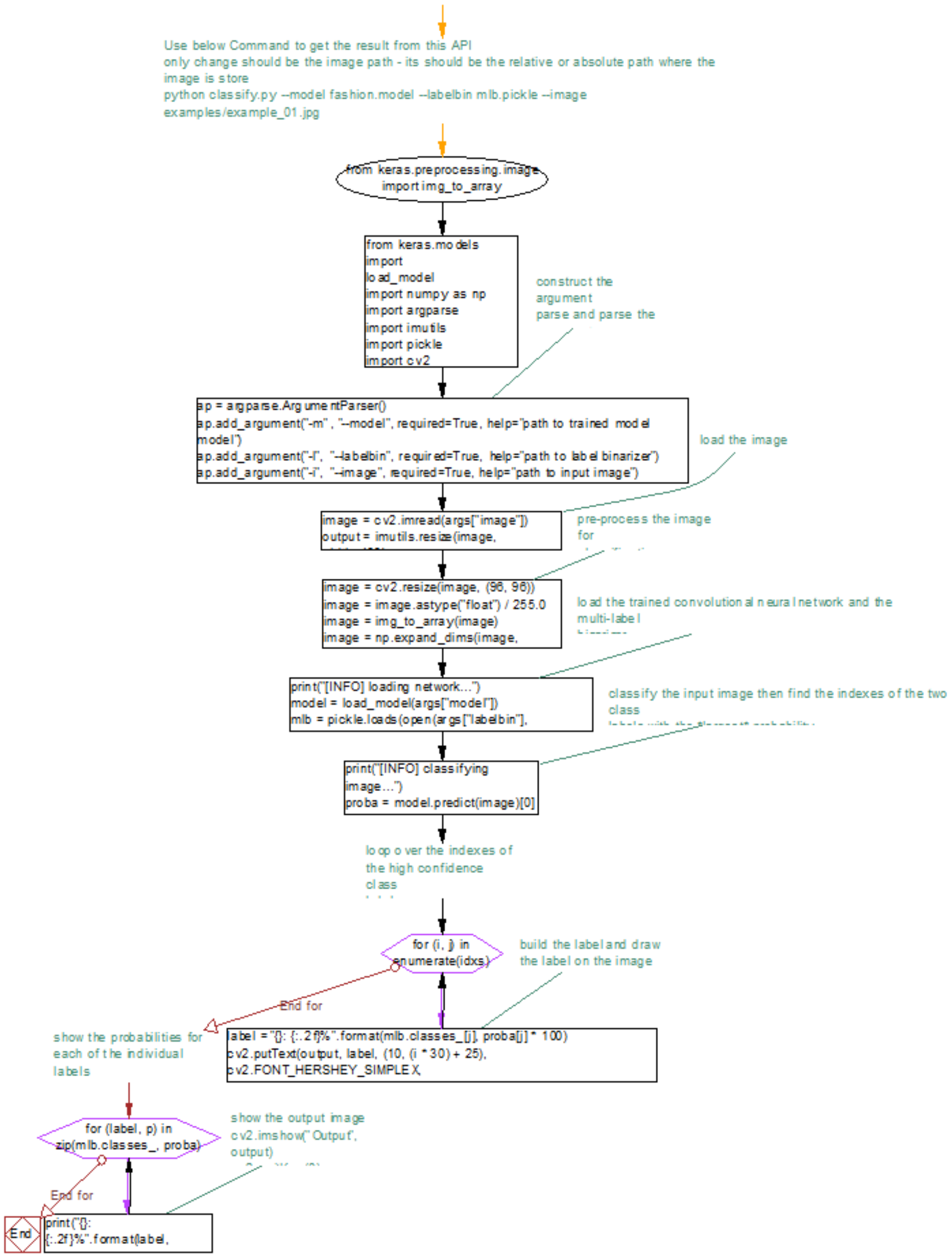
## How to test against hypotheses

Positive Scenarios(Hypothesis)

Negative Scenarios (Against Hypothesis)



## Design Document and Flowchart



Use below Command to get the result from this API
only change should be the image path - its should be the relative or absolute path where the image is store

```
python classify.py --model fashion.model --labelbin mlb.pickle --image examples/example_01.jpg
```

```
from keras.preprocessing.image import img_to_array
```

```
from keras.models import load_model
import numpy as np
import argparse
import imutils
import pickle
import cv2
```

construct the argument parse and parse the

```
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", required=True, help="path to trained model")
ap.add_argument("-l", "--labelbin", required=True, help="path to label binarizer")
ap.add_argument("-i", "--image", required=True, help="path to input image")
```

load the image

```
image = cv2.imread(args["image"])
output = imutils.resize(image,
```

pre-process the image for

```
image = cv2.resize(image, (96, 96))
image = image.astype("float") / 255.0
image = img_to_array(image)
image = np.expand_dims(image,
```

load the trained convolutional neural network and the multi-label

```
print("[INFO] loading network...")
model = load_model(args["model"])
mlb = pickle.loads(open(args["labelbin"],
```

classify the input image then find the indexes of the two class

```
print("[INFO] classifying image...")
proba = model.predict(image)[0]
```

loop over the indexes of the high confidence class

```
for (i, j) in enumerate(idxs)
```

build the label and draw the label on the image

```
label = "{}: {:.2f}%".format(mlb.classes_[j], proba[j] * 100)
cv2.putText(output, label, (10, (i * 30) + 25),
cv2.FONT_HERSHEY_SIMPLEX,
```

show the probabilities for each of the individual labels

```
for (label, p) in zip(mlb.classes_, proba)
```

show the output image

```
cv2.imshow("Output", output)
```

```
print("{}: {:.2f}%".format(label,
```

## Product Classification System Implementation

### Keras network architecture for Product classification

We import the relevant Keras modules and from there, we create our SmallerVGGNet class:

```
   # import the necessary packages
 2 from keras.models import Sequential
 3 from keras.layers.normalization import BatchNormalization
 4 from keras.layers.convolutional import Conv2D
 5 from keras.layers.convolutional import MaxPooling2D
 6 from keras.layers.core import Activation
 7 from keras.layers.core import Flatten
 8 from keras.layers.core import Dropout
 9 from keras.layers.core import Dense
10 from keras import backend as K
```

Our class is defined on Line 12. We then define the build function on Line 14, responsible for assembling the convolutional neural network

```
12 class SmallerVGGNet:
13         @staticmethod
14         def build(width, height, depth, classes, finalAct="softmax"):
15                 # initialize the model along with the input shape to be
16                 # "channels last" and the channels dimension itself
17                 model = Sequential()
18                 inputShape = (height, width, depth)
19                 chanDim = -1
20
21                 # if we are using "channels first", update the input shape
22                 # and channels dimension
23                 if K.image_data_format() == "channels_first":
24                         inputShape = (depth, height, width)
25                         chanDim = 1
```

The build method requires four parameters — width , height , depth , and classes . The depth specifies the number of channels in an input image, and classes is the number (integer) of categories/classes (not the class labels themselves). We'll use these parameters in our training script to instantiate the model with a 96 x 96 x 3 input volume.

The optional argument, finalAct (with a default value of "softmax" ) we utilized at the end of the network architecture. Changing this value from softmax to sigmoid will enable us to perform multi-label classification with Keras.

From there, we enter the body of build , initializing the model (Line 17) and defaulting to "channels_last" architecture on Lines 18 and 19 (with a convenient switch for backends that support "channels_first" architecture on Lines 23-25).

```
 7                 # CONV => RELU => POOL
28                 model.add(Conv2D(32, (3, 3), padding="same",
29                         input_shape=inputShape))
30                 model.add(Activation("relu"))
31                 model.add(BatchNormalization(axis=chanDim))
32                 model.add(MaxPooling2D(pool_size=(3, 3)))
33                 model.add(Dropout(0.25))
```

Our CONV layer has 32 filters with a 3 x 3 kernel and RELU activation (Rectified Linear Unit). We apply batch normalization, max pooling, and 25% dropout.

Dropout is the process of randomly disconnecting nodes from the current layer to the next layer. This process of random disconnects naturally helped the network to reduce overfitting as no one single node in the layer will be responsible for predicting a certain class, object, edge, or corner.

From there we have two sets of (CONV => RELU) * 2 => POOL  blocks:

```
        # (CONV => RELU) * 2 => POOL
36      model.add(Conv2D(64, (3, 3), padding="same"))
37      model.add(Activation("relu"))
38      model.add(BatchNormalization(axis=chanDim))
39      model.add(Conv2D(64, (3, 3), padding="same"))
40      model.add(Activation("relu"))
41      model.add(BatchNormalization(axis=chanDim))
42      model.add(MaxPooling2D(pool_size=(2, 2)))
43      model.add(Dropout(0.25))
44
45      # (CONV => RELU) * 2 => POOL
46      model.add(Conv2D(128, (3, 3), padding="same"))
47      model.add(Activation("relu"))
48      model.add(BatchNormalization(axis=chanDim))
49      model.add(Conv2D(128, (3, 3), padding="same"))
50      model.add(Activation("relu"))
51      model.add(BatchNormalization(axis=chanDim))
52      model.add(MaxPooling2D(pool_size=(2, 2)))
53      model.add(Dropout(0.25))
```

The changes in filters, kernels, and pool sizes in this code block which work together to progressively reduce the spatial size but increase depth.

These blocks are followed by our only set of FC => RELU  layers:

```
 5      # first (and only) set of FC => RELU layers
56      model.add(Flatten())
57      model.add(Dense(1024))
58      model.add(Activation("relu"))
59      model.add(BatchNormalization())
60      model.add(Dropout(0.5))
61
62      # use a *softmax* activation for single-label classification
63      # and *sigmoid* activation for multi-label classification
64      model.add(Dense(classes))
65      model.add(Activation(finalAct))
66
67      # return the constructed network architecture
68      return model
```

Fully connected layers are placed at the end of the network (specified by Dense on Lines 57 and 64).

Line 65 is important for our multi-label classification — finalAct dictates whether we'll use "softmax" activation for single-label classification or "sigmoid" activation

## Implementing our Keras model for multi-label product classification

Now that we have implemented SmallerVGGNet , we created train.py , the script we used to train our Keras network for multi-label classification.

```
 1 # set the matplotlib backend so figures can be saved in the background
 2 import matplotlib
 3 matplotlib.use("Agg")
 4
 5 # import the necessary packages
 6 from keras.preprocessing.image import ImageDataGenerator
 7 from keras.optimizers import Adam
 8 from keras.preprocessing.image import img_to_array
 9 from sklearn.preprocessing import MultiLabelBinarizer
10 from sklearn.model_selection import train_test_split
11 from cnn.smallervggnet import SmallerVGGNet
12 import matplotlib.pyplot as plt
13 from imutils import paths
14 import numpy as np
15 import argparse
16 import random
17 import pickle
18 import cv2
19 import os
```

On Lines 2-19 we import the packages and modules required for this script. Line 3 specifies a matplotlib backend so that we can save our plot figure in the background.

```
21 # construct the argument parse and parse the arguments
22 ap = argparse.ArgumentParser()
23 ap.add_argument("-d", "--dataset", required=True,
24               help="path to input dataset (i.e., directory of images)")
25 ap.add_argument("-m", "--model", required=True,
26               help="path to output model")
27 ap.add_argument("-l", "--labelbin", required=True,
28               help="path to output label binarizer")
29 ap.add_argument("-p", "--plot", type=str, default="plot.png",
30               help="path to output accuracy/loss plot")
31 args = vars(ap.parse_args())
```

Command line arguments to a script are like parameters to a function.

Four command line arguments (Lines 23-30):

1. --dataset : The path to our dataset.
2. --model : The path to our output serialized Keras model.
3. --labelbin : The path to our output multi-label binarizer object.
4. --plot : The path to our output plot of training loss and accuracy.

Initialized some important variables that play critical roles in our training process:

```
 3 # initialize the number of epochs to train for, initial learning rate,
34 # batch size, and image dimensions
35 EPOCHS = 75
36 INIT_LR = 1e-3
37 BS = 32
38 IMAGE_DIMS = (96, 96, 3)
```

These variables on Lines 35-38 define that:

- Our network was trained for 75 EPOCHS in order to learn patterns by incremental improvements via backpropagation.

- We established an initial learning rate of 1e-3 (the default value for the Adam optimizer).
- The batch size is 32 . we adjusted this value depending on your CPU capability but we found a batch size of 32 works well for this project.
- As stated above, our images are 96 x 96 and contain 3 channels.

From there, the next two code blocks handle loading and preprocessing our training data:

```
# grab the image paths and randomly shuffle them
print("[INFO] loading images...")
imagePaths = sorted(list(paths.list_images(args["dataset"])))
random.seed(42)
random.shuffle(imagePaths)

# initialize the data and labels
data = []
labels = []
```

Here we grabbed the imagePaths and shuffling them randomly, followed by initializing data and labels lists.

Next, we looped over the imagePaths , preprocess the image data, and extract multi-class-labels.

```
# loop over the input images
for imagePath in imagePaths:
	# load the image, pre-process it, and store it in the data list
	image = cv2.imread(imagePath)
	image = cv2.resize(image, (IMAGE_DIMS[1], IMAGE_DIMS[0]))
	image = img_to_array(image)
	data.append(image)

	# extract set of class labels from the image path and update the
	# labels list
	l = label = imagePath.split(os.path.sep)[-2].split("_")
	labels.append(l)
```

First, we load each image into memory (Line 53). Then, we perform preprocessing on Lines 54 and 55. We appended the image to data (Line 56).

Lines 60 and 61 handles splitting the image path into multiple labels for our multi-label classification task. After Line 60 is executed, a 2-element list is created and is then appended to the labels list on Line 61.

```
$ python
>>> import os
>>> labels = []
>>> imagePath = "dataset/red_dress/long_dress_from_macys_red.png"
>>> l = label = imagePath.split(os.path.sep)[-2].split("_")
>>> l
['red', 'dress']
>>> labels.append(l)
>>>
>>> imagePath = "dataset/blue_jeans/stylish_blue_jeans_from_your_favorite_store.png"
>>> l = label = imagePath.split(os.path.sep)[-2].split("_")
>>> labels.append(l)
```

```
>>>
>>> imagePath = "dataset/red_shirt/red_shirt_from_target.png"
>>> l = label = imagePath.split(os.path.sep)[-2].split("_")
>>> labels.append(l)
>>>
>>> labels
[['red', 'dress'], ['blue', 'jeans'], ['red', 'shirt']]
```

The labels list is a “list of lists” — each element of labels is a 2-element list. The two labels for each list is constructed based on the file path of the input image.

```
# scale the raw pixel intensities to the range [0, 1]
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)
print("[INFO] data matrix: {} images ({:.2f}MB)".format(
        len(imagePaths), data.nbytes / (1024 * 1000.0)))
```

Our data list contains images stored as NumPy arrays. In a single line of code, we converted the list to a NumPy array and scale the pixel intensities to the range [0, 1] .

We also converted labels to a NumPy array as well.

From there, let’s binarize the labels — the below block is critical for this week’s multi-class classification concept:

```
# binarize the labels using scikit-learn's special multi-label
# binarizer implementation
print("[INFO] class labels:")
mlb = MultiLabelBinarizer()
labels = mlb.fit_transform(labels)

# loop over each of the possible class labels and show them
for (i, label) in enumerate(mlb.classes_):
        print("{}. {}".format(i + 1, label))
```

MultiLabelBinarizer transforms a tuple of ("red", "dress”) to a vector with six total categories:

```
$ python
>>> from sklearn.preprocessing import MultiLabelBinarizer
>>> labels = [
...     ("blue", "jeans"),
...     ("blue", "dress"),
...     ("red", "dress"),
...     ("red", "shirt"),
...     ("blue", "shirt"),
...     ("black", "jeans")
... ]
>>> mlb = MultiLabelBinarizer()
>>> mlb.fit(labels)
MultiLabelBinarizer(classes=None, sparse_output=False)
>>> mlb.classes_
array(['black', 'blue', 'dress', 'jeans', 'red', 'shirt'], dtype=object)
>>> mlb.transform([("red", "dress")])
array([[0, 0, 1, 0, 1, 0]])
```

Following we constructed, the training and testing splits as well as initialize the data augmenter:

```
   # partition the data into training and testing splits using 80% of
80 # the data for training and the remaining 20% for testing
81 (trainX, testX, trainY, testY) = train_test_split(data,
82             labels, test_size=0.2, random_state=42)
83
84 # construct the image generator for data augmentation
85 aug = ImageDataGenerator(rotation_range=25, width_shift_range=0.1,
86             height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
87             horizontal_flip=True, fill_mode="nearest")
```

Splitting the data for training and testing is common in machine learning practice — We've allocated 80% of the images for training data and 20% for testing data. This is handled by scikit-learn on Lines 81 and 82.

Our data augmenter object is initialized on Lines 85-87. Data augmentation is a best practice and a most-likely a "must" if you are working with less than 1,000 images per class.

Next, let's build the model and initialize the Adam optimizer:

```
   # initialize the model using a sigmoid activation as the final layer
90 # in the network so we can perform multi-label classification
91 print("[INFO] compiling model...")
92 model = SmallerVGGNet.build(
93             width=IMAGE_DIMS[1], height=IMAGE_DIMS[0],
94             depth=IMAGE_DIMS[2], classes=len(mlb.classes_),
95             finalAct="sigmoid")
96
97 # initialize the optimizer
98 opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
```

On Lines 92-95 we build our SmallerVGGNet model, noting the finalAct="sigmoid" parameter indicating that we'll be performing multi-label classification.

From there, we are compiling the model and kick off training:

```
    # compile the model using binary cross-entropy rather than
101 # categorical cross-entropy -- this may seem counterintuitive for
102 # multi-label classification
103
104
105 model.compile(loss="binary_crossentropy", optimizer=opt,
106             metrics=["accuracy"])
107
108 # train the network
109 print("[INFO] training network...")
110 H = model.fit_generator(
111             aug.flow(trainX, trainY, batch_size=BS),
112             validation_data=(testX, testY),
113             steps_per_epoch=len(trainX) // BS,
114             epochs=EPOCHS, verbose=1)
```

On Lines 105 and 106 we compile the model using binary cross-entropy rather than categorical cross-entropy.

This may seem counterintuitive for multi-label classification; however, the goal was to treat each output label as an independent Bernoulli distribution and we wanted to penalize each output node independently.

From there we launch the training process with our data augmentation generator (Lines 110-114).

After training is complete we can save our model and label binarizer to disk:

```
116 # save the model to disk
117 print("[INFO] serializing network...")
118 model.save(args["model"])
119
120 # save the multi-label binarizer to disk
121 print("[INFO] serializing label binarizer...")
122 f = open(args["labelbin"], "wb")
123 f.write(pickle.dumps(mlb))
124 f.close()
```

we plotted accuracy and loss:

```
126 # plot the training loss and accuracy
127 plt.style.use("ggplot")
128 plt.figure()
129 N = EPOCHS
130 plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
131 plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
132 plt.plot(np.arange(0, N), H.history["acc"], label="train_acc")
133 plt.plot(np.arange(0, N), H.history["val_acc"], label="val_acc")
134 plt.title("Training Loss and Accuracy")
135 plt.xlabel("Epoch #")
136 plt.ylabel("Loss/Accuracy")
137 plt.legend(loc="upper left")
138 plt.savefig(args["plot"])
```

Accuracy + loss for training and validation is plotted on Lines 127-137. The plot is saved as an image file on Line 138.

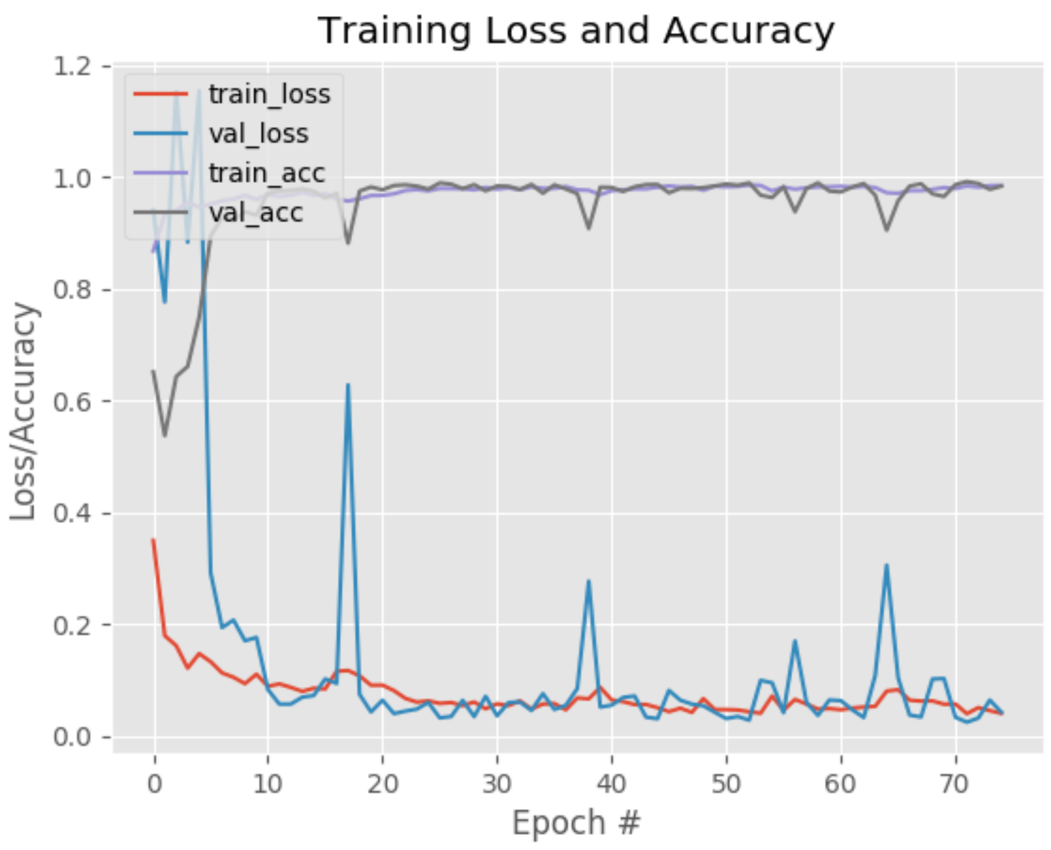
## Training a Keras network for multi-label classification

```
$ python train.py --dataset dataset --model fashion.model \
        --labelbin mlb.pickle
Using TensorFlow backend.
[INFO] loading images...
[INFO] data matrix: 2165 images (467.64MB)
[INFO] class labels:
1. black
2. blue
3. dress
4. jeans
5. red
6. shirt
[INFO] compiling model...
[INFO] training network...
Epoch 1/75
name: Macbook Pro
54/54 [==============================] - 4s - loss: 0.3503 - acc: 0.8682 - val_loss: 0.9417 - val_acc: 0.6520
Epoch 2/75
54/54 [==============================] - 2s - loss: 0.1833 - acc: 0.9324 - val_loss: 0.7770 - val_acc: 0.5377
```

```
Epoch 3/75
54/54 [==============================] - 2s - loss: 0.1736 - acc: 0.9378 - val_loss: 1.1532 - val_acc: 0.6436
...
Epoch 73/75
54/54 [==============================] - 2s - loss: 0.0534 - acc: 0.9813 - val_loss: 0.0324 - val_acc: 0.9888
Epoch 74/75
54/54 [==============================] - 2s - loss: 0.0518 - acc: 0.9833 - val_loss: 0.0645 - val_acc: 0.9784
Epoch 75/75
54/54 [==============================] - 2s - loss: 0.0405 - acc: 0.9857 - val_loss: 0.0429 - val_acc: 0.9842
[INFO] serializing network...
[INFO] serializing label binarizer...
```

we trained the network for 75 epochs, achieving:

- 98.57% multi-label classification accuracy on the training set
- 98.42% multi-label classification accuracy on the testing set



## Applied Keras multi-label classification to new images

On Lines 2-9 we import the necessary packages for this script. We used Keras and OpenCV in this script.

```
   # import the necessary packages
 2 from keras.preprocessing.image import img_to_array
 3 from keras.models import load_model
 4 import numpy as np
 5 import argparse
 6 import imutils
 7 import pickle
 8 import cv2
 9 import os
10
11 # construct the argument parse and parse the arguments
12 ap = argparse.ArgumentParser()
13 ap.add_argument("-m", "--model", required=True,
14 	help="path to trained model model")
15 ap.add_argument("-l", "--labelbin", required=True,
16 	help="path to label binarizer")
17 ap.add_argument("-i", "--image", required=True,
18 	help="path to input image")
19 args = vars(ap.parse_args())
```

Then we proceeded to parse our three required command line arguments on Lines 12-19.

From there, we load and pre-process the input image:

```
 1 # load the image
22 image = cv2.imread(args["image"])
23 output = imutils.resize(image, width=400)
24
25 # pre-process the image for classification
26 image = cv2.resize(image, (96, 96))
27 image = image.astype("float") / 255.0
28 image = img_to_array(image)
29 image = np.expand_dims(image, axis=0)


   # load the trained convolutional neural network and the multi-label
32 # binarizer
33 print("[INFO] loading network...")
34 model = load_model(args["model"])
35 mlb = pickle.loads(open(args["labelbin"], "rb").read())
36
37 # classify the input image then find the indexes of the two class
38 # labels with the *largest* probability
39 print("[INFO] classifying image...")
40 proba = model.predict(image)[0]
41 idxs = np.argsort(proba)[::-1][:2]
```

Following we prepared the class labels + associated confidence values for overlay on the output image:

```
   # loop over the indexes of the high confidence class labels
45 for (i, j) in enumerate(idxs):
46 	# build the label and draw the label on the image
47 	label = "{}: {:.2f}%".format(mlb.classes_[j], proba[j] * 100)
48 	cv2.putText(output, label, (10, (i * 30) + 25),
49 		cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
50
51 # show the probabilities for each of the individual labels
52 for (label, p) in zip(mlb.classes_, proba):
53 	print("{}: {:.2f}%".format(label, p * 100))
54
55 # show the output image
56 cv2.imshow("Output", output)
   cv2.waitKey(0)
```

The loop on Lines 44-48 draws the top two multi-label predictions and corresponding confidence values on the output image.

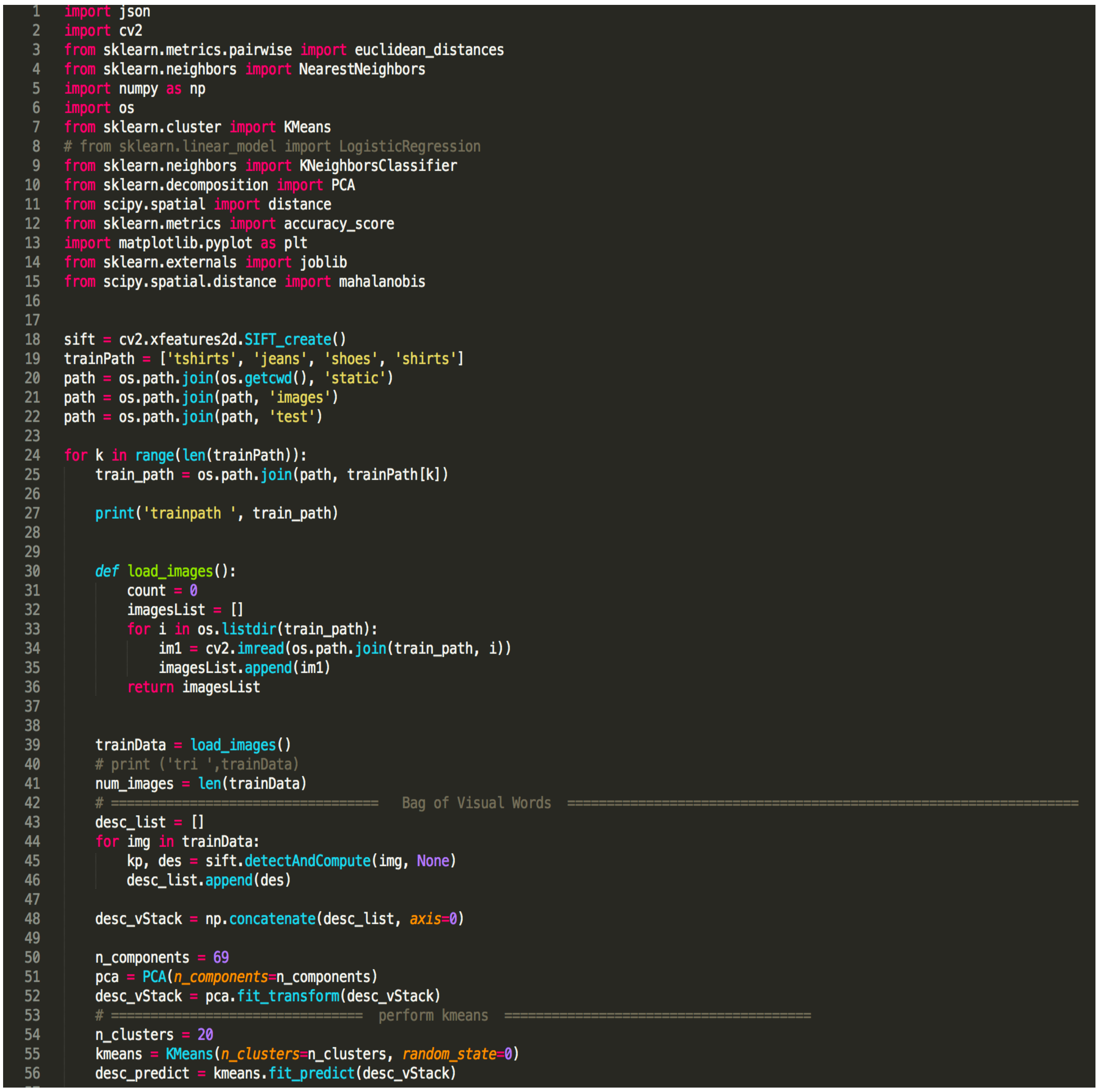
Finally, we show the output image on the screen (Lines 55 and 56).

### Command to get the classification output:

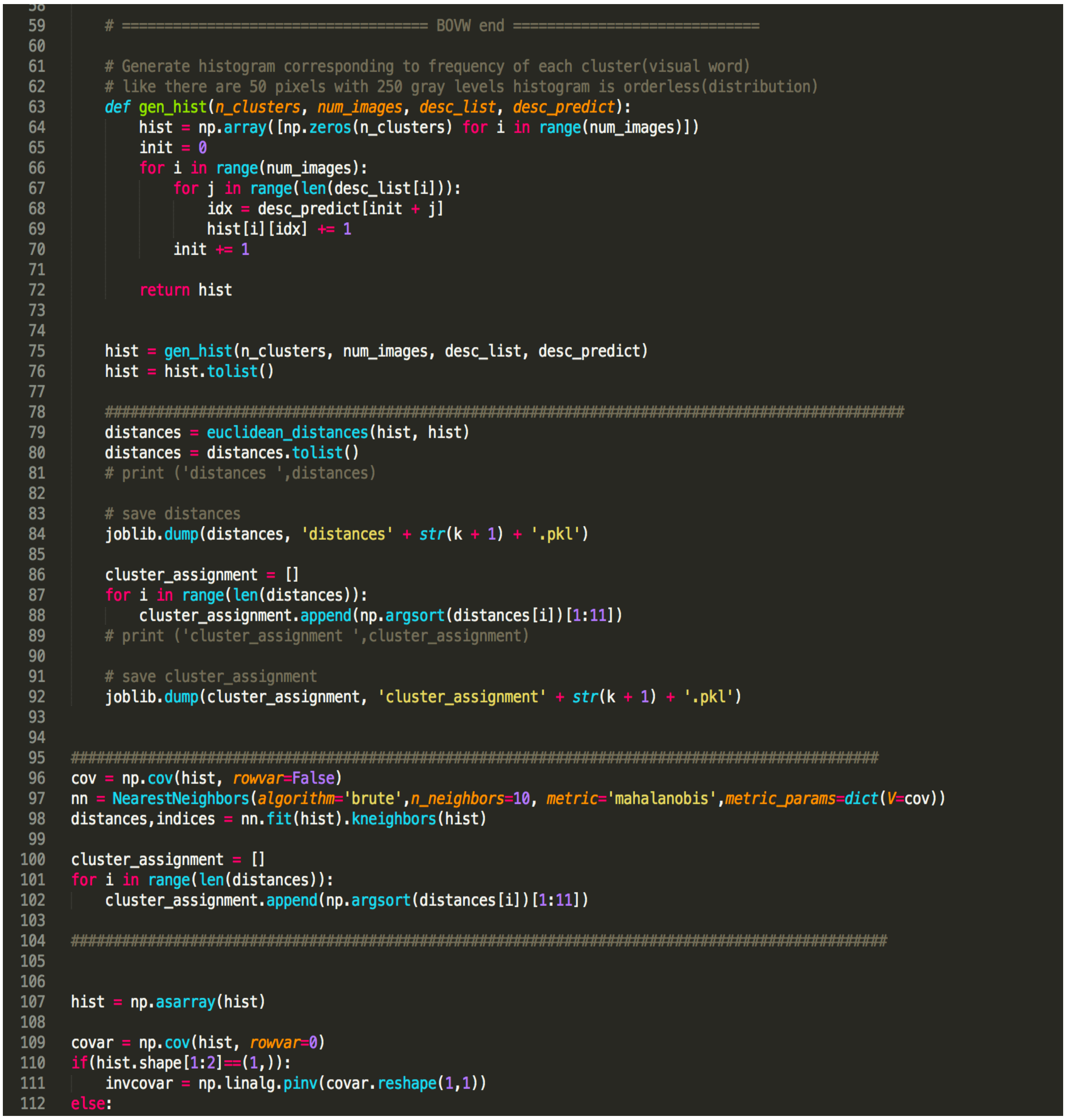
python classify.py --model fashion.model --labelbin mlb.pickle  --image \
examples/example_01.jpg

## Recommendation System Implementation

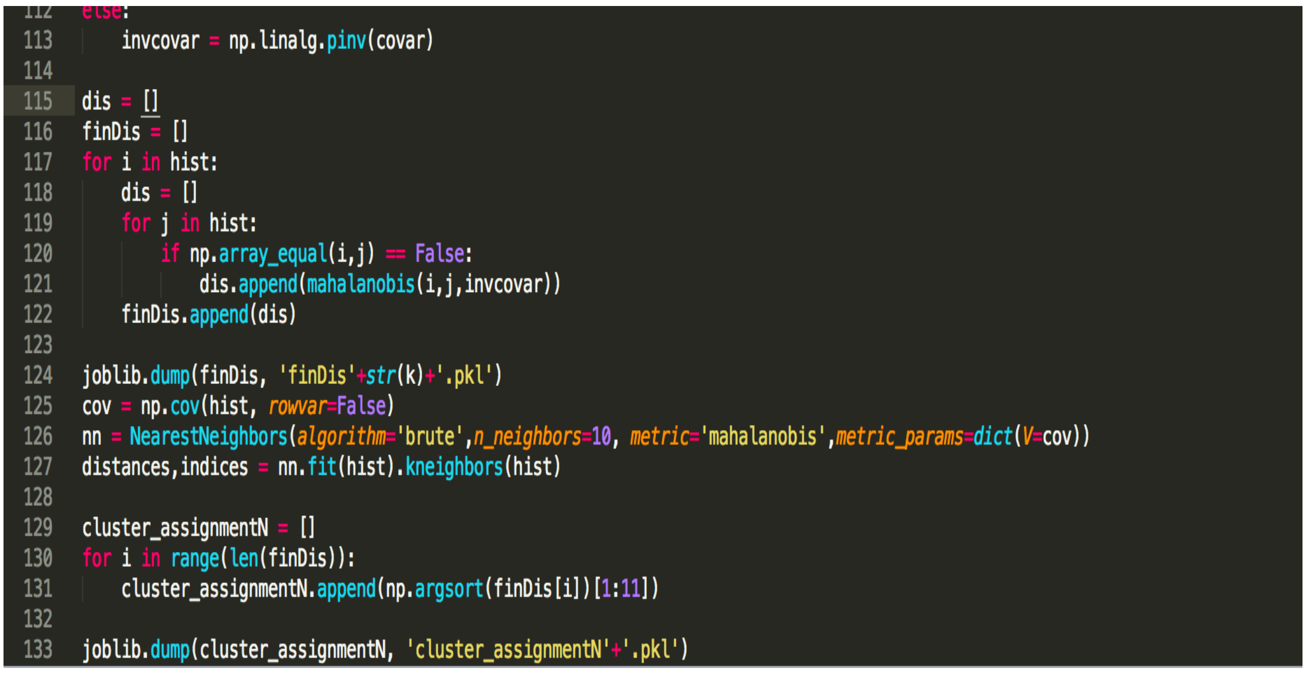
To create biased clusters with set of training data



```python
import json
import cv2
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.neighbors import NearestNeighbors
import numpy as np
import os
from sklearn.cluster import KMeans
# from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from scipy.spatial import distance
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.externals import joblib
from scipy.spatial.distance import mahalanobis


sift = cv2.xfeatures2d.SIFT_create()
trainPath = ['tshirts', 'jeans', 'shoes', 'shirts']
path = os.path.join(os.getcwd(), 'static')
path = os.path.join(path, 'images')
path = os.path.join(path, 'test')

for k in range(len(trainPath)):
    train_path = os.path.join(path, trainPath[k])

    print('trainpath ', train_path)


    def load_images():
        count = 0
        imagesList = []
        for i in os.listdir(train_path):
            im1 = cv2.imread(os.path.join(train_path, i))
            imagesList.append(im1)
        return imagesList


    trainData = load_images()
    # print ('tri ',trainData)
    num_images = len(trainData)
    # ==================================   Bag of Visual Words   =================================================================
    desc_list = []
    for img in trainData:
        kp, des = sift.detectAndCompute(img, None)
        desc_list.append(des)

    desc_vStack = np.concatenate(desc_list, axis=0)

    n_components = 69
    pca = PCA(n_components=n_components)
    desc_vStack = pca.fit_transform(desc_vStack)
    # ================================   perform kmeans   ======================================
    n_clusters = 20
    kmeans = KMeans(n_clusters=n_clusters, random_state=0)
    desc_predict = kmeans.fit_predict(desc_vStack)
```

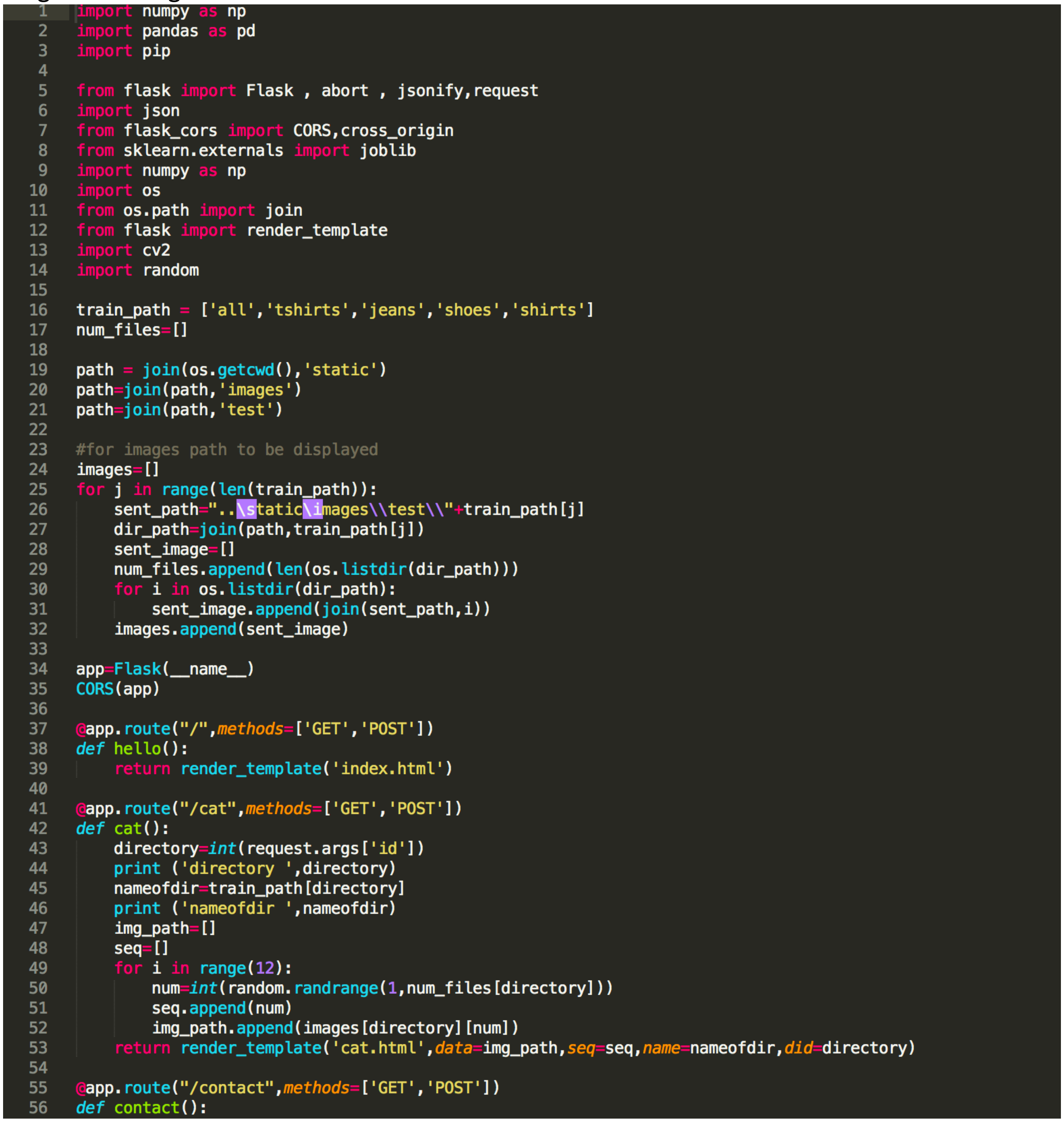


```
    # ==================================== BOVW end ==============================

    # Generate histogram corresponding to frequency of each cluster(visual word)
    # like there are 50 pixels with 250 gray levels histogram is orderless(distribution)
    def gen_hist(n_clusters, num_images, desc_list, desc_predict):
        hist = np.array([np.zeros(n_clusters) for i in range(num_images)])
        init = 0
        for i in range(num_images):
            for j in range(len(desc_list[i])):
                idx = desc_predict[init + j]
                hist[i][idx] += 1
            init += 1

        return hist


    hist = gen_hist(n_clusters, num_images, desc_list, desc_predict)
    hist = hist.tolist()

    ###############################################################################################
    distances = euclidean_distances(hist, hist)
    distances = distances.tolist()
    # print ('distances ',distances)

    # save distances
    joblib.dump(distances, 'distances' + str(k + 1) + '.pkl')

    cluster_assignment = []
    for i in range(len(distances)):
        cluster_assignment.append(np.argsort(distances[i])[1:11])
    # print ('cluster_assignment ',cluster_assignment)

    # save cluster_assignment
    joblib.dump(cluster_assignment, 'cluster_assignment' + str(k + 1) + '.pkl')


###############################################################################################
cov = np.cov(hist, rowvar=False)
nn = NearestNeighbors(algorithm='brute',n_neighbors=10, metric='mahalanobis',metric_params=dict(V=cov))
distances,indices = nn.fit(hist).kneighbors(hist)

cluster_assignment = []
for i in range(len(distances)):
    cluster_assignment.append(np.argsort(distances[i])[1:11])

###############################################################################################


hist = np.asarray(hist)

covar = np.cov(hist, rowvar=0)
if(hist.shape[1:2]==(1,)):
    invcovar = np.linalg.pinv(covar.reshape(1,1))
else:
```



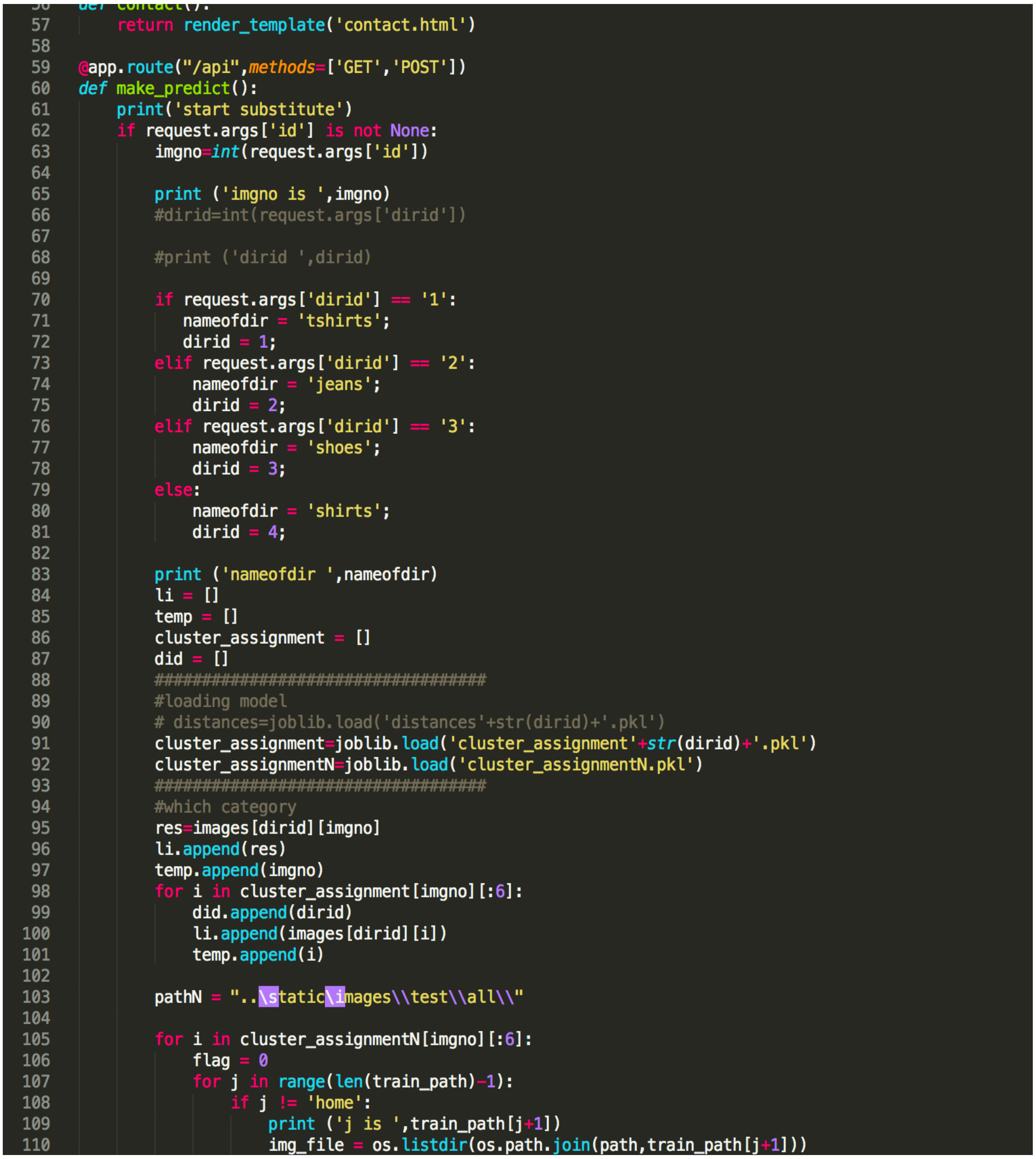
```
else:
    invcovar = np.linalg.pinv(covar)

dis = []
finDis = []
for i in hist:
    dis = []
    for j in hist:
        if np.array_equal(i,j) == False:
            dis.append(mahalanobis(i,j,invcovar))
    finDis.append(dis)

joblib.dump(finDis, 'finDis'+str(k)+'.pkl')
cov = np.cov(hist, rowvar=False)
nn = NearestNeighbors(algorithm='brute',n_neighbors=10, metric='mahalanobis',metric_params=dict(V=cov))
distances,indices = nn.fit(hist).kneighbors(hist)

cluster_assignmentN = []
for i in range(len(finDis)):
    cluster_assignmentN.append(np.argsort(finDis[i])[1:11])

joblib.dump(cluster_assignmentN, 'cluster_assignmentN'+'.pkl')
```

To get the images from the

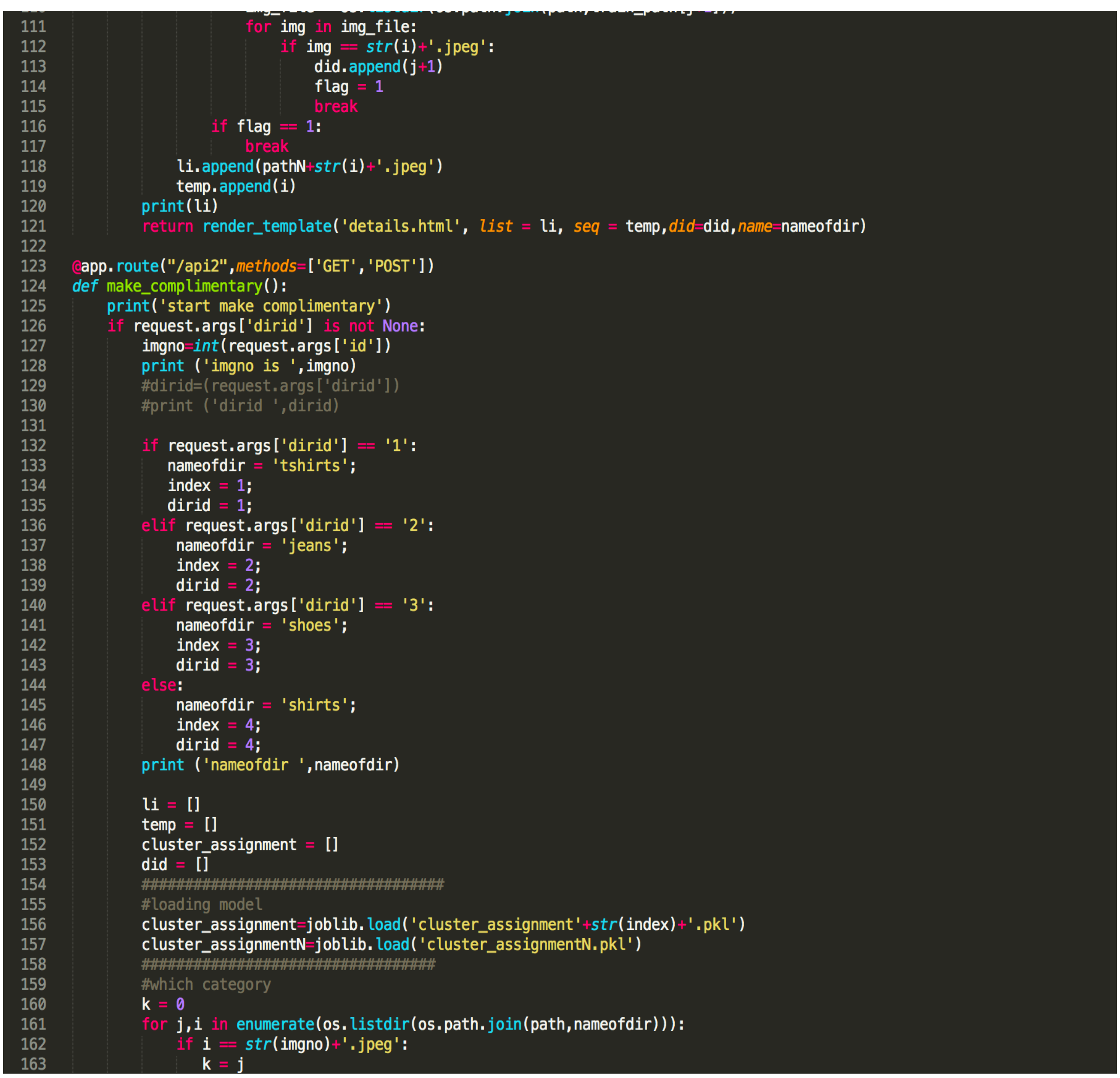


```
import numpy as np
import pandas as pd
import pip

from flask import Flask , abort , jsonify,request
import json
from flask_cors import CORS,cross_origin
from sklearn.externals import joblib
import numpy as np
import os
from os.path import join
from flask import render_template
import cv2
import random

train_path = ['all','tshirts','jeans','shoes','shirts']
num_files=[]

path = join(os.getcwd(),'static')
path=join(path,'images')
path=join(path,'test')

#for images path to be displayed
images=[]
for j in range(len(train_path)):
    sent_path="..\static\images\\test\\"+train_path[j]
    dir_path=join(path,train_path[j])
    sent_image=[]
    num_files.append(len(os.listdir(dir_path)))
    for i in os.listdir(dir_path):
        sent_image.append(join(sent_path,i))
    images.append(sent_image)

app=Flask(__name__)
CORS(app)

@app.route("/",methods=['GET','POST'])
def hello():
    return render_template('index.html')

@app.route("/cat",methods=['GET','POST'])
def cat():
    directory=int(request.args['id'])
    print ('directory ',directory)
    nameofdir=train_path[directory]
    print ('nameofdir ',nameofdir)
    img_path=[]
    seq=[]
    for i in range(12):
        num=int(random.randrange(1,num_files[directory]))
        seq.append(num)
        img_path.append(images[directory][num])
    return render_template('cat.html',data=img_path,seq=seq,name=nameofdir,did=directory)

@app.route("/contact",methods=['GET','POST'])
def contact():
```

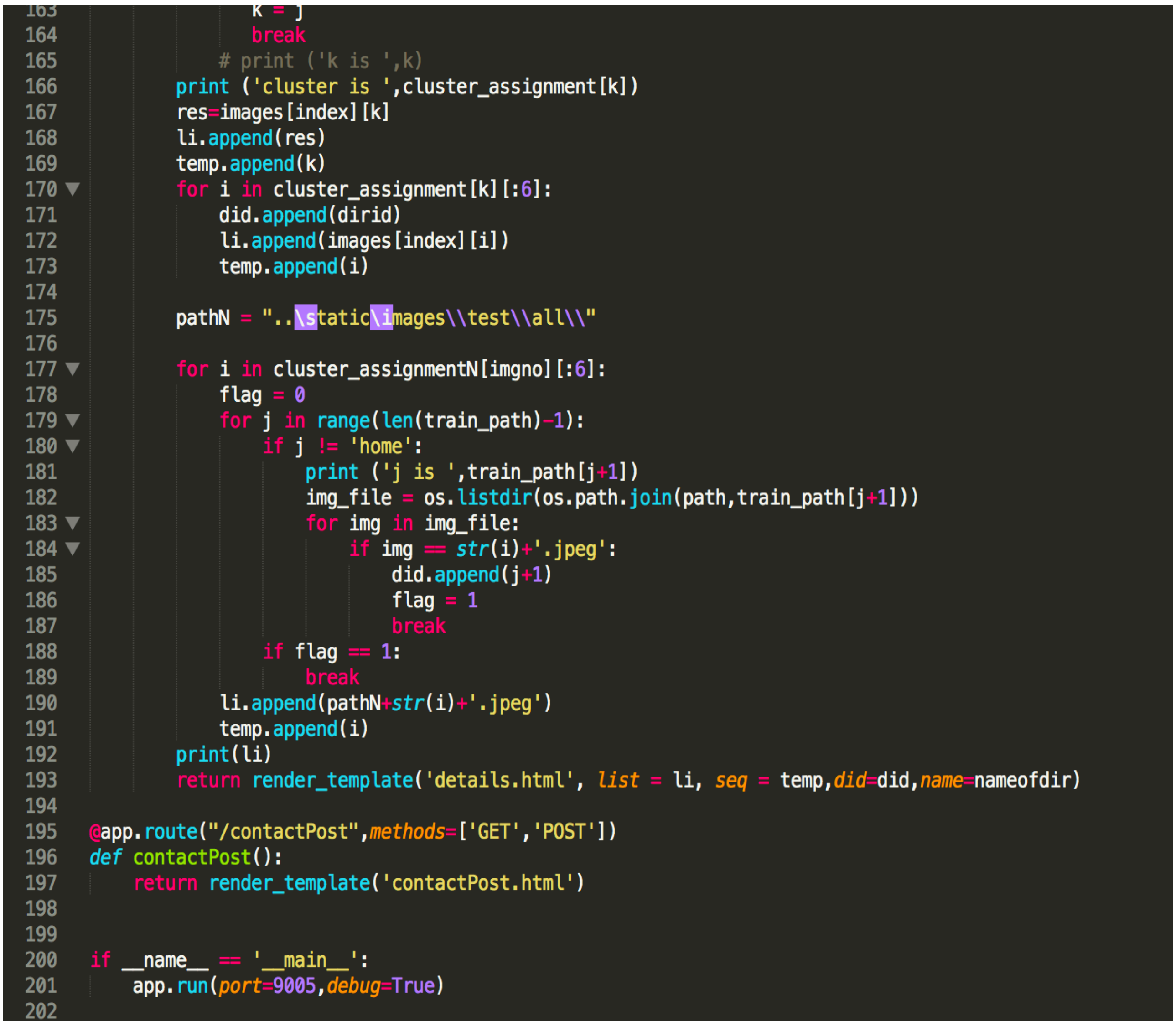
cluster and send it to the UI.



```
def contact():
    return render_template('contact.html')

@app.route("/api",methods=['GET','POST'])
def make_predict():
    print('start substitute')
    if request.args['id'] is not None:
        imgno=int(request.args['id'])

        print ('imgno is ',imgno)
        #dirid=int(request.args['dirid'])

        #print ('dirid ',dirid)

        if request.args['dirid'] == '1':
            nameofdir = 'tshirts';
            dirid = 1;
        elif request.args['dirid'] == '2':
            nameofdir = 'jeans';
            dirid = 2;
        elif request.args['dirid'] == '3':
            nameofdir = 'shoes';
            dirid = 3;
        else:
            nameofdir = 'shirts';
            dirid = 4;

        print ('nameofdir ',nameofdir)
        li = []
        temp = []
        cluster_assignment = []
        did = []
        ###################################
        #loading model
        # distances=joblib.load('distances'+str(dirid)+'.pkl')
        cluster_assignment=joblib.load('cluster_assignment'+str(dirid)+'.pkl')
        cluster_assignmentN=joblib.load('cluster_assignmentN.pkl')
        ###################################
        #which category
        res=images[dirid][imgno]
        li.append(res)
        temp.append(imgno)
        for i in cluster_assignment[imgno][:6]:
            did.append(dirid)
            li.append(images[dirid][i])
            temp.append(i)

        pathN = "..\static\images\\test\\all\\"

        for i in cluster_assignmentN[imgno][:6]:
            flag = 0
            for j in range(len(train_path)-1):
                if j != 'home':
                    print ('j is ',train_path[j+1])
                    img_file = os.listdir(os.path.join(path,train_path[j+1]))
```

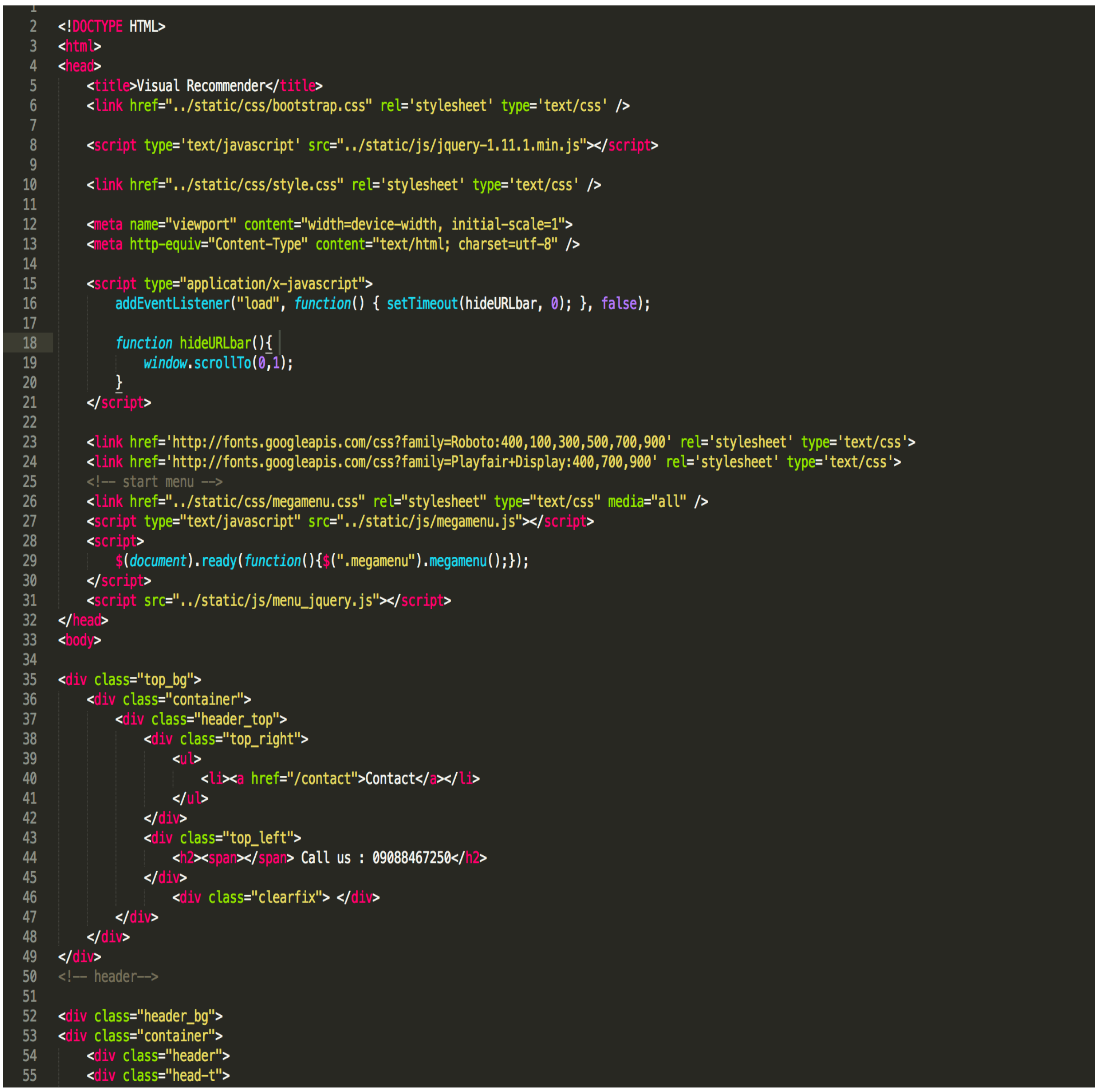


```
                    for img in img_file:
                        if img == str(i)+'.jpeg':
                            did.append(j+1)
                            flag = 1
                            break
                if flag == 1:
                    break
            li.append(pathN+str(i)+'.jpeg')
            temp.append(i)
        print(li)
        return render_template('details.html', list = li, seq = temp,did=did,name=nameofdir)

@app.route("/api2",methods=['GET','POST'])
def make_complimentary():
    print('start make complimentary')
    if request.args['dirid'] is not None:
        imgno=int(request.args['id'])
        print ('imgno is ',imgno)
        #dirid=(request.args['dirid'])
        #print ('dirid ',dirid)

        if request.args['dirid'] == '1':
           nameofdir = 'tshirts';
           index = 1;
           dirid = 1;
        elif request.args['dirid'] == '2':
            nameofdir = 'jeans';
            index = 2;
            dirid = 2;
        elif request.args['dirid'] == '3':
            nameofdir = 'shoes';
            index = 3;
            dirid = 3;
        else:
            nameofdir = 'shirts';
            index = 4;
            dirid = 4;
        print ('nameofdir ',nameofdir)

        li = []
        temp = []
        cluster_assignment = []
        did = []
        ###################################
        #loading model
        cluster_assignment=joblib.load('cluster_assignment'+str(index)+'.pkl')
        cluster_assignmentN=joblib.load('cluster_assignmentN.pkl')
        ##################################
        #which category
        k = 0
        for j,i in enumerate(os.listdir(os.path.join(path,nameofdir))):
            if i == str(imgno)+'.jpeg':
               k = j
```

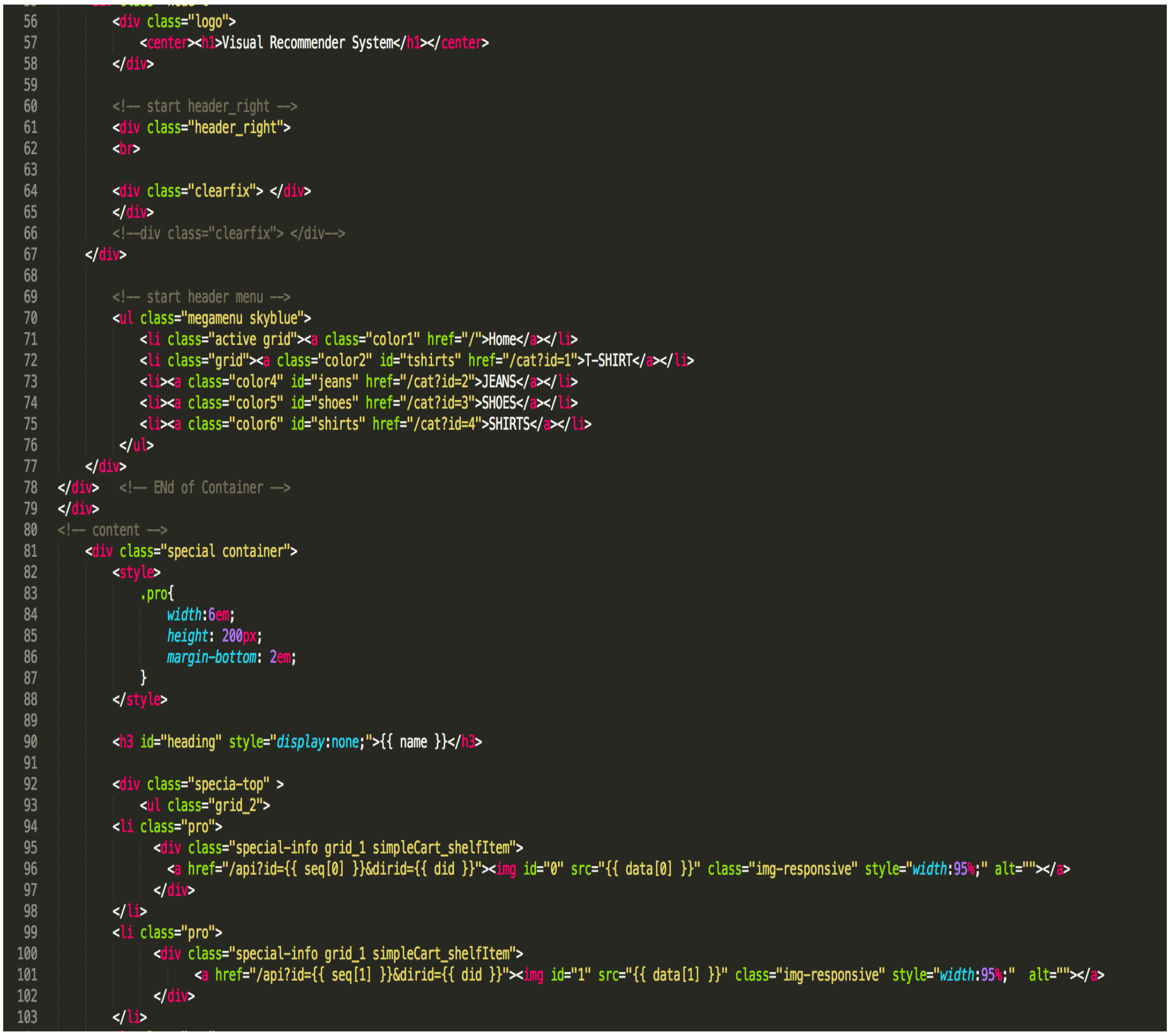


```
                k = j
                break
            # print ('k is ',k)
        print ('cluster is ',cluster_assignment[k])
        res=images[index][k]
        li.append(res)
        temp.append(k)
        for i in cluster_assignment[k][:6]:
            did.append(dirid)
            li.append(images[index][i])
            temp.append(i)

        pathN = "..\static\images\\test\\all\\"

        for i in cluster_assignmentN[imgno][:6]:
            flag = 0
            for j in range(len(train_path)-1):
                if j != 'home':
                    print ('j is ',train_path[j+1])
                    img_file = os.listdir(os.path.join(path,train_path[j+1]))
                    for img in img_file:
                        if img == str(i)+'.jpeg':
                            did.append(j+1)
                            flag = 1
                            break
                if flag == 1:
                    break
            li.append(pathN+str(i)+'.jpeg')
            temp.append(i)
        print(li)
        return render_template('details.html', list = li, seq = temp,did=did,name=nameofdir)

@app.route("/contactPost",methods=['GET','POST'])
def contactPost():
    return render_template('contactPost.html')


if __name__ == '__main__':
    app.run(port=9005,debug=True)
```

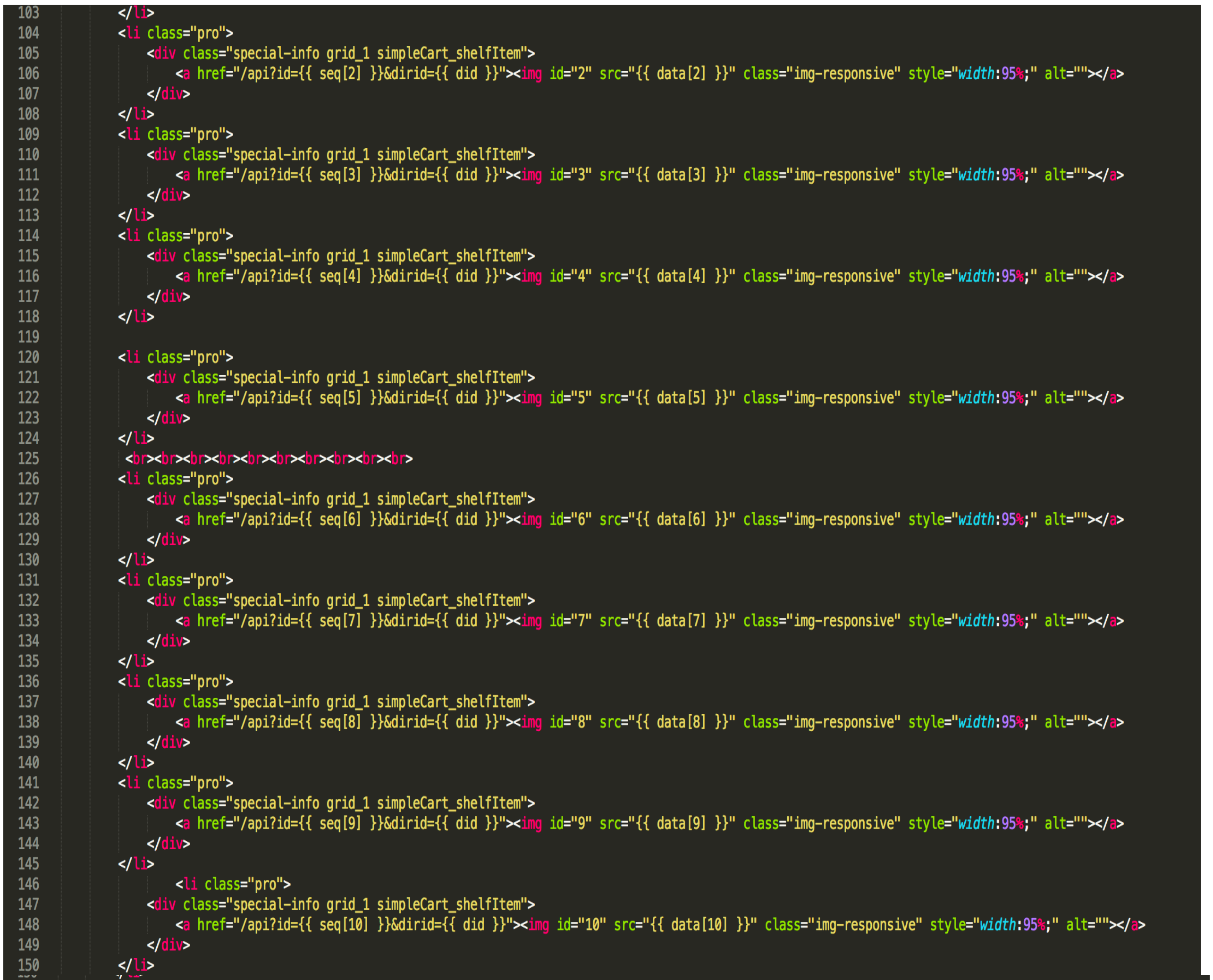
Get the index from the python code and display on the UI.



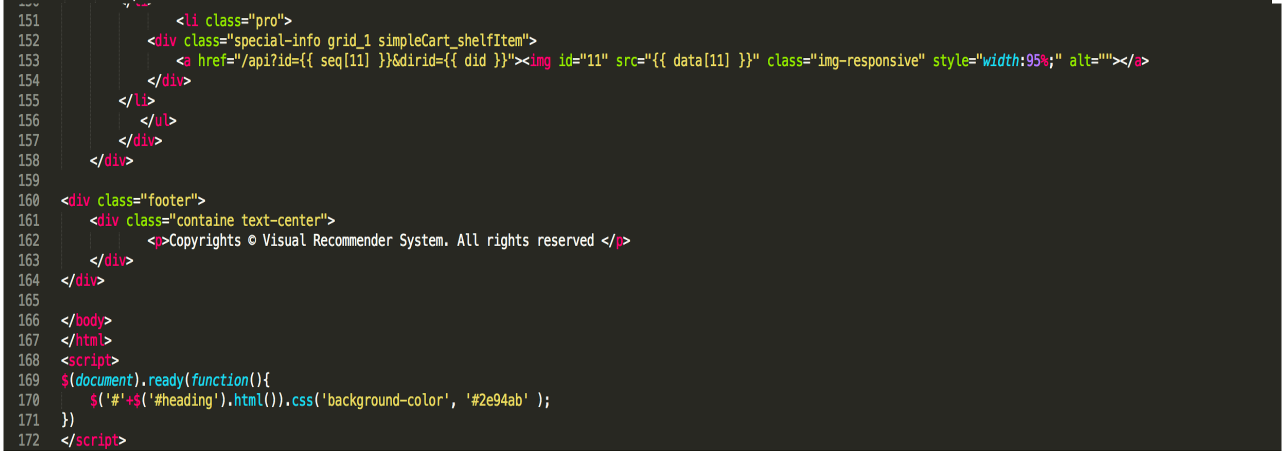
```
<!DOCTYPE HTML>
<html>
<head>
    <title>Visual Recommender</title>
    <link href="../static/css/bootstrap.css" rel='stylesheet' type='text/css' />

    <script type='text/javascript' src="../static/js/jquery-1.11.1.min.js"></script>

    <link href="../static/css/style.css" rel='stylesheet' type='text/css' />

    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

    <script type="application/x-javascript">
        addEventListener("load", function() { setTimeout(hideURLbar, 0); }, false);

        function hideURLbar(){
            window.scrollTo(0,1);
        }
    </script>

    <link href='http://fonts.googleapis.com/css?family=Roboto:400,100,300,500,700,900' rel='stylesheet' type='text/css'>
    <link href='http://fonts.googleapis.com/css?family=Playfair+Display:400,700,900' rel='stylesheet' type='text/css'>
    <!-- start menu -->
    <link href="../static/css/megamenu.css" rel="stylesheet" type="text/css" media="all" />
    <script type="text/javascript" src="../static/js/megamenu.js"></script>
    <script>
        $(document).ready(function(){$(".megamenu").megamenu();});
    </script>
    <script src="../static/js/menu_jquery.js"></script>
</head>
<body>

<div class="top_bg">
    <div class="container">
        <div class="header_top">
            <div class="top_right">
                <ul>
                    <li><a href="/contact">Contact</a></li>
                </ul>
            </div>
            <div class="top_left">
                <h2><span></span> Call us : 09088467250</h2>
            </div>
                <div class="clearfix"> </div>
        </div>
    </div>
</div>
<!-- header-->

<div class="header_bg">
<div class="container">
    <div class="header">
    <div class="head-t">
```



```
        <div class="logo">
            <center><h1>Visual Recommender System</h1></center>
        </div>

        <!-- start header_right -->
        <div class="header_right">
        <br>

        <div class="clearfix"> </div>
        </div>
        <!--div class="clearfix"> </div-->
    </div>

        <!-- start header menu -->
        <ul class="megamenu skyblue">
            <li class="active grid"><a class="color1" href="/">Home</a></li>
            <li class="grid"><a class="color2" id="tshirts" href="/cat?id=1">T-SHIRT</a></li>
            <li><a class="color4" id="jeans" href="/cat?id=2">JEANS</a></li>
            <li><a class="color5" id="shoes" href="/cat?id=3">SHOES</a></li>
            <li><a class="color6" id="shirts" href="/cat?id=4">SHIRTS</a></li>
         </ul>
    </div>
</div>   <!-- ENd of Container -->
</div>
<!-- content -->
    <div class="special container">
        <style>
            .pro{
                width:6em;
                height: 200px;
                margin-bottom: 2em;
            }
        </style>

        <h3 id="heading" style="display:none;">{{ name }}</h3>

        <div class="specia-top" >
            <ul class="grid_2">
        <li class="pro">
              <div class="special-info grid_1 simpleCart_shelfItem">
                <a href="/api?id={{ seq[0] }}&dirid={{ did }}"><img id="0" src="{{ data[0] }}" class="img-responsive" style="width:95%;" alt=""></a>
              </div>
        </li>
        <li class="pro">
              <div class="special-info grid_1 simpleCart_shelfItem">
                    <a href="/api?id={{ seq[1] }}&dirid={{ did }}"><img id="1" src="{{ data[1] }}" class="img-responsive" style="width:95%;"  alt=""></a>
              </div>
        </li>
```

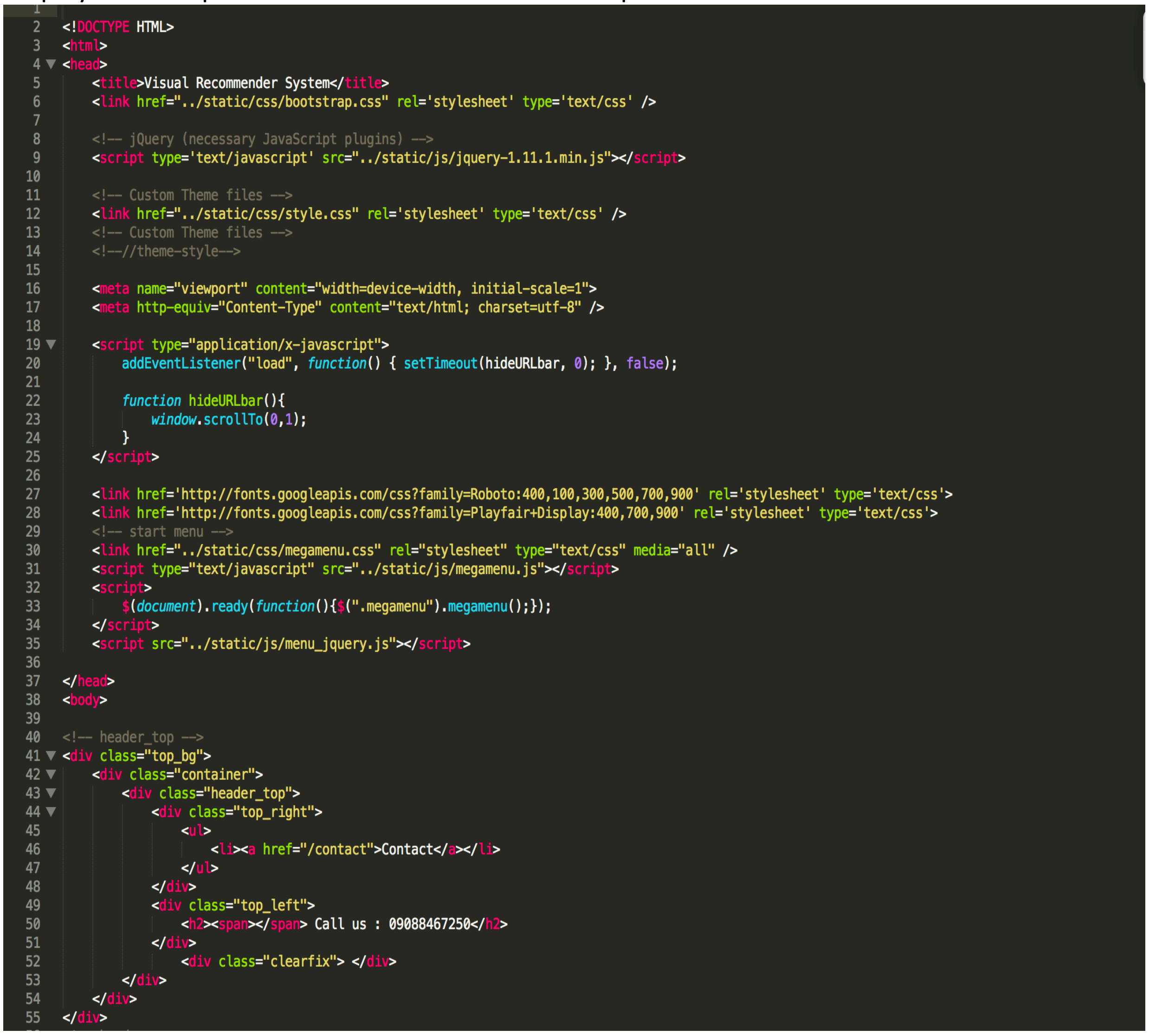


```
        </li>
        <li class="pro">
            <div class="special-info grid_1 simpleCart_shelfItem">
                <a href="/api?id={{ seq[2] }}&dirid={{ did }}"><img id="2" src="{{ data[2] }}" class="img-responsive" style="width:95%;" alt=""></a>
            </div>
        </li>
        <li class="pro">
            <div class="special-info grid_1 simpleCart_shelfItem">
                <a href="/api?id={{ seq[3] }}&dirid={{ did }}"><img id="3" src="{{ data[3] }}" class="img-responsive" style="width:95%;" alt=""></a>
            </div>
        </li>
        <li class="pro">
            <div class="special-info grid_1 simpleCart_shelfItem">
                <a href="/api?id={{ seq[4] }}&dirid={{ did }}"><img id="4" src="{{ data[4] }}" class="img-responsive" style="width:95%;" alt=""></a>
            </div>
        </li>

        <li class="pro">
            <div class="special-info grid_1 simpleCart_shelfItem">
                <a href="/api?id={{ seq[5] }}&dirid={{ did }}"><img id="5" src="{{ data[5] }}" class="img-responsive" style="width:95%;" alt=""></a>
            </div>
        </li>
         <br><br><br><br><br><br><br><br><br><br>
        <li class="pro">
            <div class="special-info grid_1 simpleCart_shelfItem">
                <a href="/api?id={{ seq[6] }}&dirid={{ did }}"><img id="6" src="{{ data[6] }}" class="img-responsive" style="width:95%;" alt=""></a>
            </div>
        </li>
        <li class="pro">
            <div class="special-info grid_1 simpleCart_shelfItem">
                <a href="/api?id={{ seq[7] }}&dirid={{ did }}"><img id="7" src="{{ data[7] }}" class="img-responsive" style="width:95%;" alt=""></a>
            </div>
        </li>
        <li class="pro">
            <div class="special-info grid_1 simpleCart_shelfItem">
                <a href="/api?id={{ seq[8] }}&dirid={{ did }}"><img id="8" src="{{ data[8] }}" class="img-responsive" style="width:95%;" alt=""></a>
            </div>
        </li>
        <li class="pro">
            <div class="special-info grid_1 simpleCart_shelfItem">
                <a href="/api?id={{ seq[9] }}&dirid={{ did }}"><img id="9" src="{{ data[9] }}" class="img-responsive" style="width:95%;" alt=""></a>
            </div>
        </li>
                <li class="pro">
            <div class="special-info grid_1 simpleCart_shelfItem">
                <a href="/api?id={{ seq[10] }}&dirid={{ did }}"><img id="10" src="{{ data[10] }}" class="img-responsive" style="width:95%;" alt=""></a>
            </div>
        </li>
```

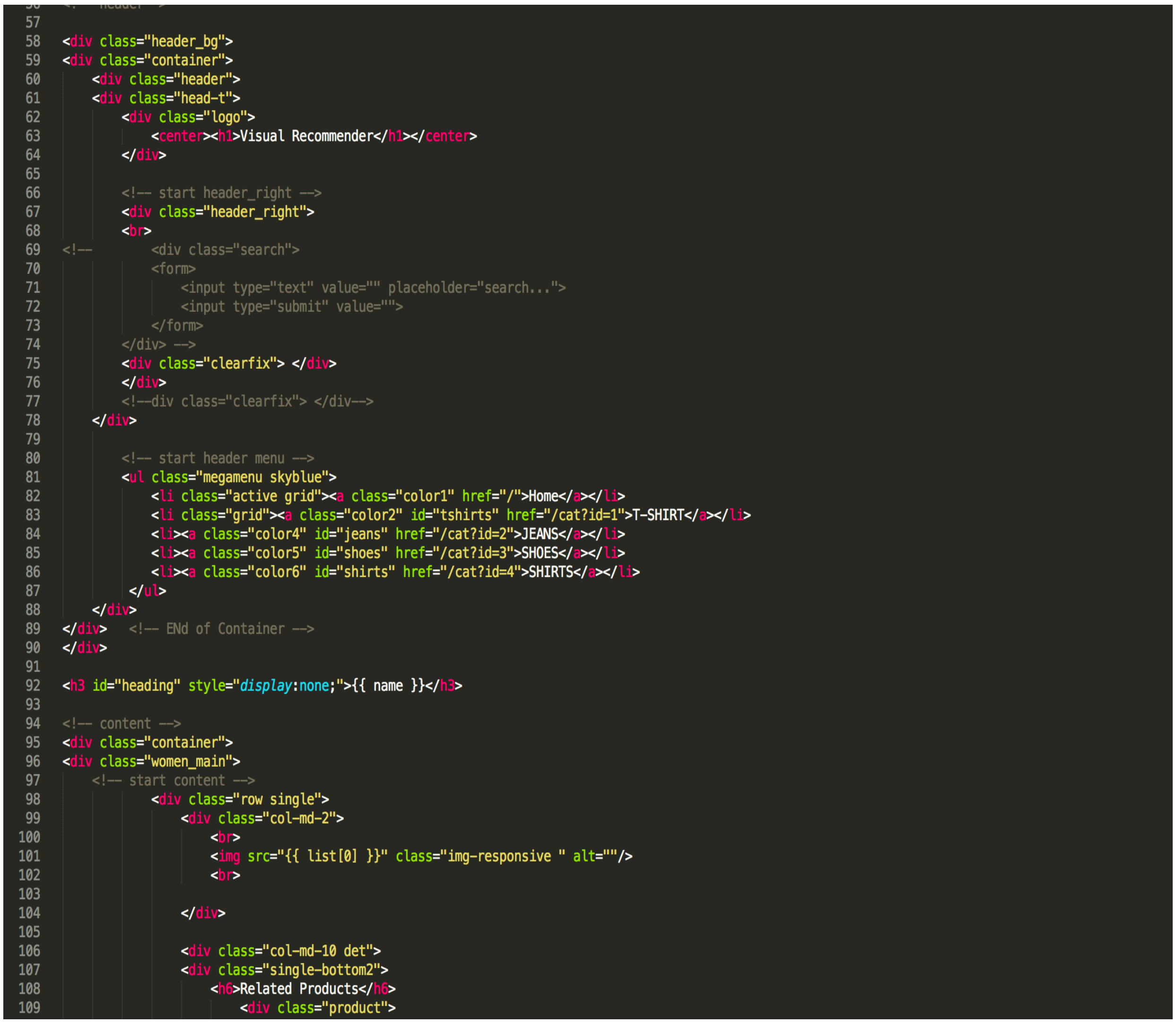


```
                <li class="pro">
            <div class="special-info grid_1 simpleCart_shelfItem">
                <a href="/api?id={{ seq[11] }}&dirid={{ did }}"><img id="11" src="{{ data[11] }}" class="img-responsive" style="width:95%;" alt=""></a>
            </div>
        </li>
          </ul>
        </div>
    </div>

<div class="footer">
    <div class="containe text-center">
            <p>Copyrights © Visual Recommender System. All rights reserved </p>
    </div>
</div>

</body>
</html>
<script>
$(document).ready(function(){
    $('#'+$('#heading').html()).css('background-color', '#2e94ab' );
})
</script>
```

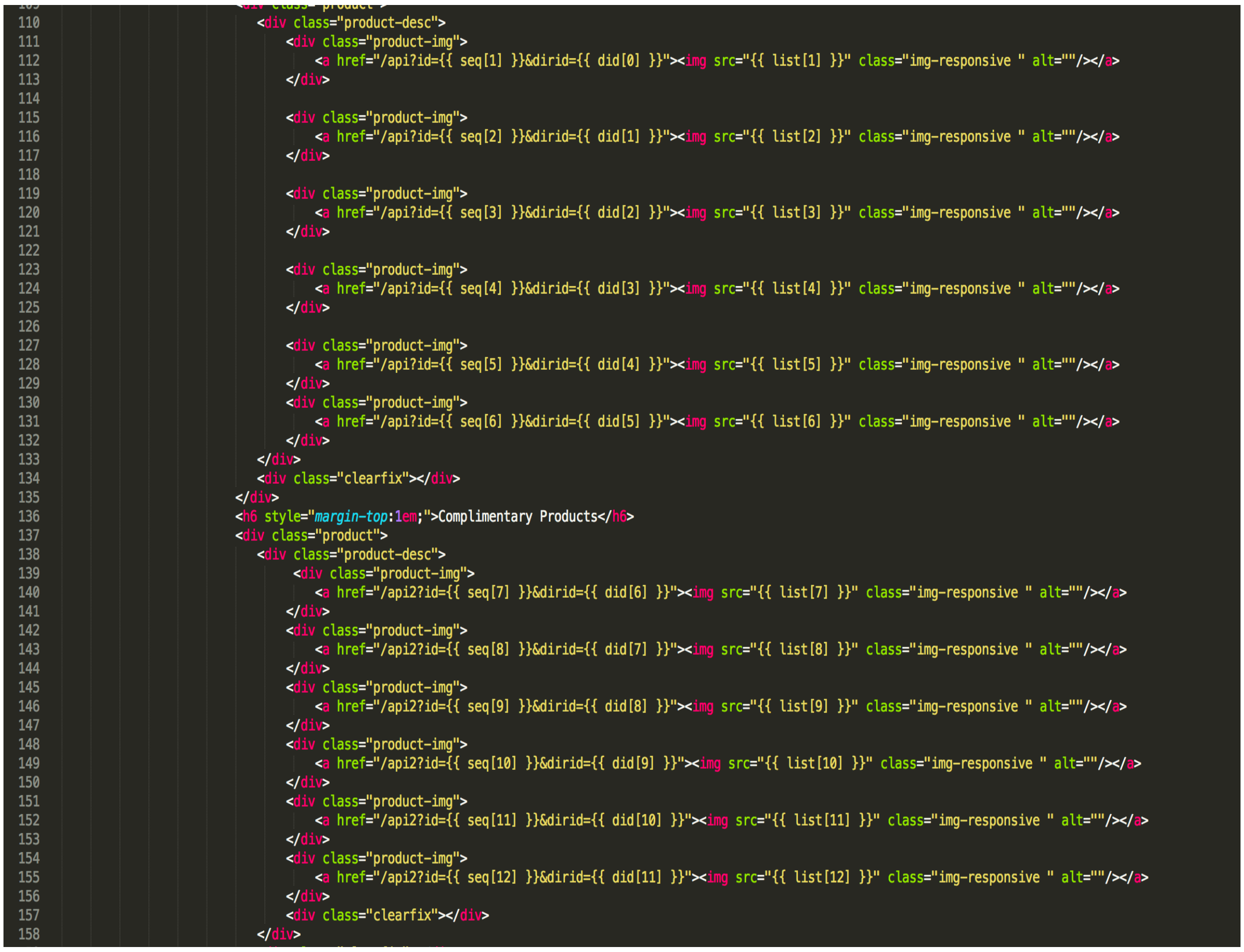
Displays related products as well as recommended product.



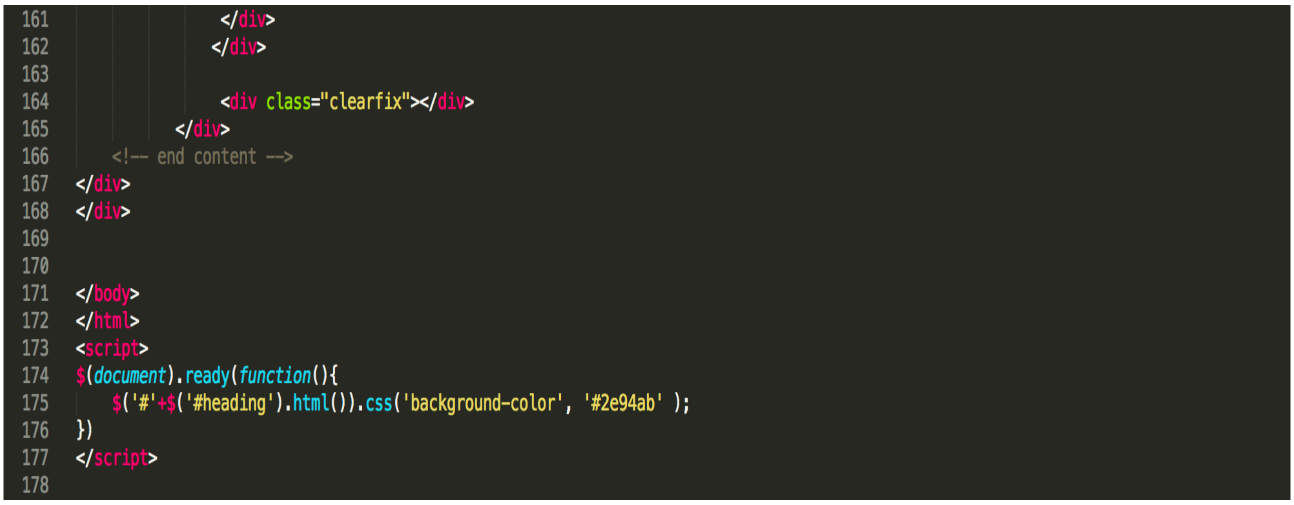
```
<!DOCTYPE HTML>
<html>
<head>
    <title>Visual Recommender System</title>
    <link href="../static/css/bootstrap.css" rel='stylesheet' type='text/css' />

    <!-- jQuery (necessary JavaScript plugins) -->
    <script type='text/javascript' src="../static/js/jquery-1.11.1.min.js"></script>

    <!-- Custom Theme files -->
    <link href="../static/css/style.css" rel='stylesheet' type='text/css' />
    <!-- Custom Theme files -->
    <!--//theme-style-->

    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

    <script type="application/x-javascript">
        addEventListener("load", function() { setTimeout(hideURLbar, 0); }, false);

        function hideURLbar(){
            window.scrollTo(0,1);
        }
    </script>

    <link href='http://fonts.googleapis.com/css?family=Roboto:400,100,300,500,700,900' rel='stylesheet' type='text/css'>
    <link href='http://fonts.googleapis.com/css?family=Playfair+Display:400,700,900' rel='stylesheet' type='text/css'>
    <!-- start menu -->
    <link href="../static/css/megamenu.css" rel="stylesheet" type="text/css" media="all" />
    <script type="text/javascript" src="../static/js/megamenu.js"></script>
    <script>
        $(document).ready(function(){$(".megamenu").megamenu();});
    </script>
    <script src="../static/js/menu_jquery.js"></script>

</head>
<body>

<!-- header_top -->
<div class="top_bg">
    <div class="container">
        <div class="header_top">
            <div class="top_right">
                <ul>
                    <li><a href="/contact">Contact</a></li>
                </ul>
            </div>
            <div class="top_left">
                <h2><span></span> Call us : 09088467250</h2>
            </div>
                <div class="clearfix"> </div>
        </div>
    </div>
</div>
```



```html

<div class="header_bg">
<div class="container">
    <div class="header">
    <div class="head-t">
        <div class="logo">
            <center><h1>Visual Recommender</h1></center>
        </div>

        <!-- start header_right -->
        <div class="header_right">
        <br>
<!--        <div class="search">
            <form>
                <input type="text" value="" placeholder="search...">
                <input type="submit" value="">
            </form>
        </div> -->
        <div class="clearfix"> </div>
        </div>
        <!--div class="clearfix"> </div-->
    </div>

        <!-- start header menu -->
        <ul class="megamenu skyblue">
            <li class="active grid"><a class="color1" href="/">Home</a></li>
            <li class="grid"><a class="color2" id="tshirts" href="/cat?id=1">T-SHIRT</a></li>
            <li><a class="color4" id="jeans" href="/cat?id=2">JEANS</a></li>
            <li><a class="color5" id="shoes" href="/cat?id=3">SHOES</a></li>
            <li><a class="color6" id="shirts" href="/cat?id=4">SHIRTS</a></li>
         </ul>
    </div>
</div>   <!-- ENd of Container -->
</div>

<h3 id="heading" style="display:none;">{{ name }}</h3>

<!-- content -->
<div class="container">
<div class="women_main">
    <!-- start content -->
            <div class="row single">
                <div class="col-md-2">
                    <br>
                    <img src="{{ list[0] }}" class="img-responsive " alt=""/>
                    <br>

                </div>

                <div class="col-md-10 det">
                <div class="single-bottom2">
                    <h6>Related Products</h6>
                        <div class="product">
```

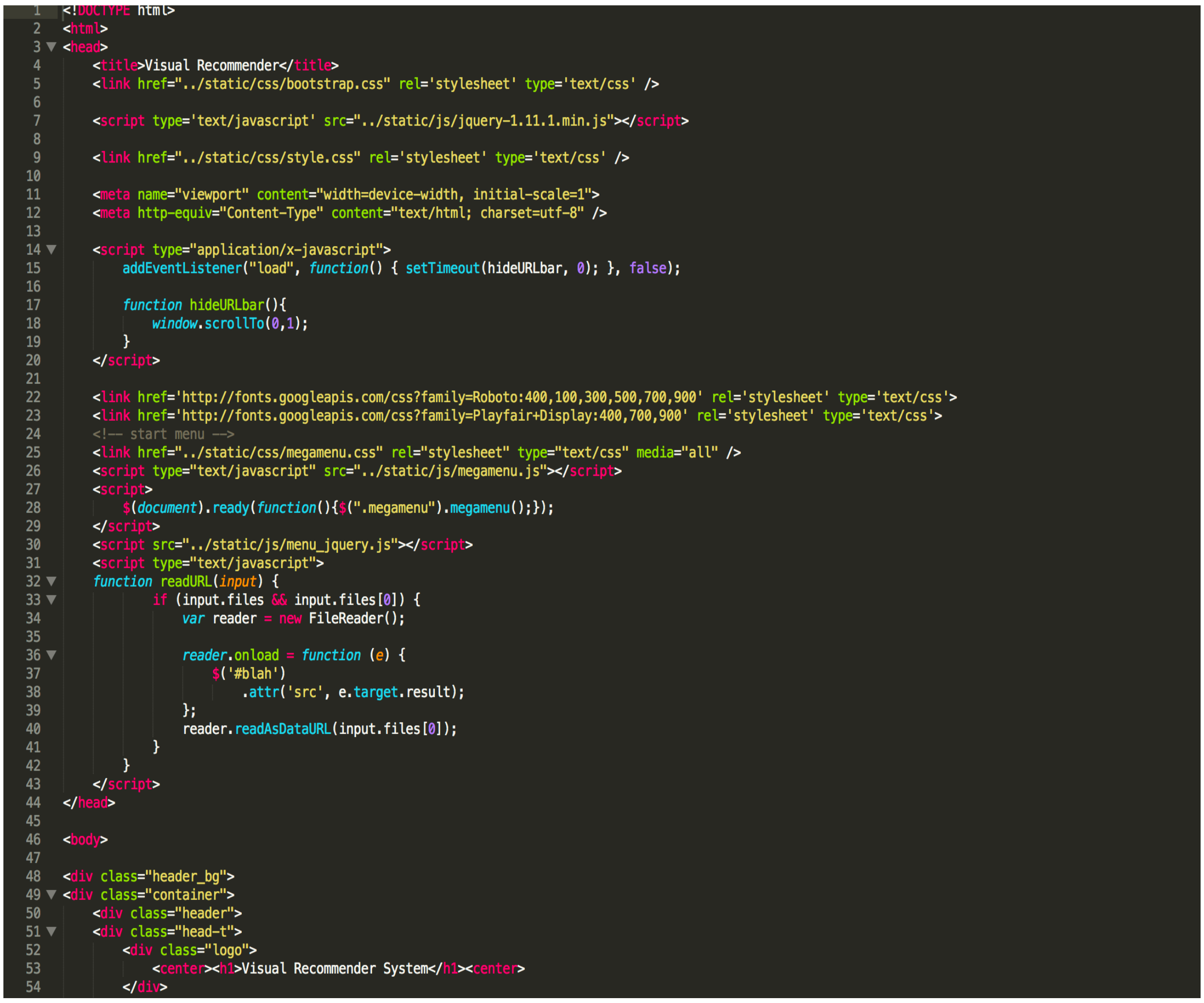


```
    <div class="product-desc">
        <div class="product-img">
            <a href="/api?id={{ seq[1] }}&dirid={{ did[0] }}"><img src="{{ list[1] }}" class="img-responsive " alt=""/></a>
        </div>

        <div class="product-img">
            <a href="/api?id={{ seq[2] }}&dirid={{ did[1] }}"><img src="{{ list[2] }}" class="img-responsive " alt=""/></a>
        </div>

        <div class="product-img">
            <a href="/api?id={{ seq[3] }}&dirid={{ did[2] }}"><img src="{{ list[3] }}" class="img-responsive " alt=""/></a>
        </div>

        <div class="product-img">
            <a href="/api?id={{ seq[4] }}&dirid={{ did[3] }}"><img src="{{ list[4] }}" class="img-responsive " alt=""/></a>
        </div>

        <div class="product-img">
            <a href="/api?id={{ seq[5] }}&dirid={{ did[4] }}"><img src="{{ list[5] }}" class="img-responsive " alt=""/></a>
        </div>
        <div class="product-img">
            <a href="/api?id={{ seq[6] }}&dirid={{ did[5] }}"><img src="{{ list[6] }}" class="img-responsive " alt=""/></a>
        </div>
    </div>
    <div class="clearfix"></div>
</div>
<h6 style="margin-top:1em;">Complimentary Products</h6>
<div class="product">
    <div class="product-desc">
         <div class="product-img">
            <a href="/api2?id={{ seq[7] }}&dirid={{ did[6] }}"><img src="{{ list[7] }}" class="img-responsive " alt=""/></a>
        </div>
        <div class="product-img">
            <a href="/api2?id={{ seq[8] }}&dirid={{ did[7] }}"><img src="{{ list[8] }}" class="img-responsive " alt=""/></a>
        </div>
        <div class="product-img">
            <a href="/api2?id={{ seq[9] }}&dirid={{ did[8] }}"><img src="{{ list[9] }}" class="img-responsive " alt=""/></a>
        </div>
        <div class="product-img">
            <a href="/api2?id={{ seq[10] }}&dirid={{ did[9] }}"><img src="{{ list[10] }}" class="img-responsive " alt=""/></a>
        </div>
        <div class="product-img">
            <a href="/api2?id={{ seq[11] }}&dirid={{ did[10] }}"><img src="{{ list[11] }}" class="img-responsive " alt=""/></a>
        </div>
        <div class="product-img">
            <a href="/api2?id={{ seq[12] }}&dirid={{ did[11] }}"><img src="{{ list[12] }}" class="img-responsive " alt=""/></a>
        </div>
        <div class="clearfix"></div>
    </div>
```

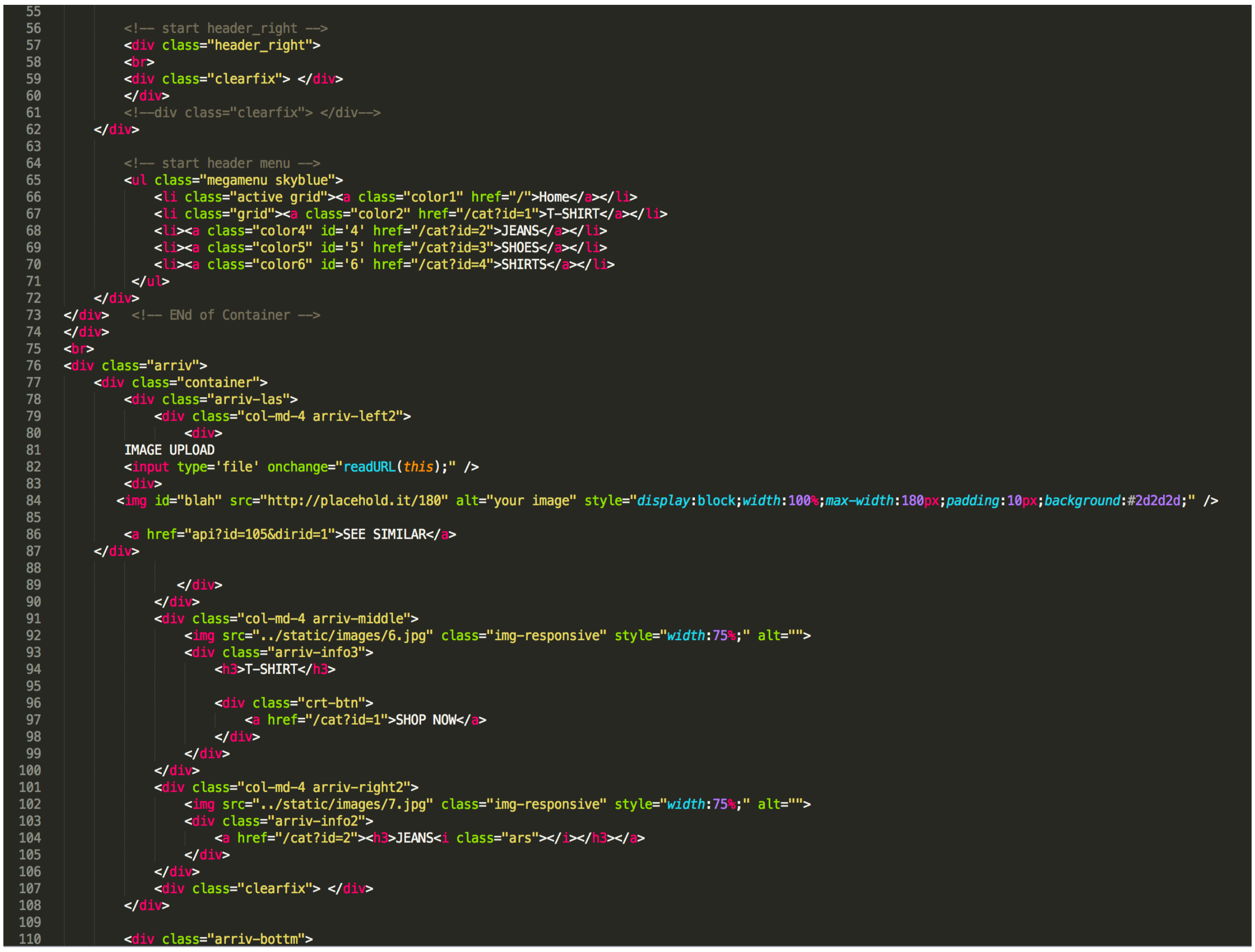


```
               </div>
              </div>

               <div class="clearfix"></div>
          </div>
    <!-- end content -->
</div>
</div>


</body>
</html>
<script>
$(document).ready(function(){
    $('#'+$('#heading').html()).css('background-color', '#2e94ab' );
})
</script>
```

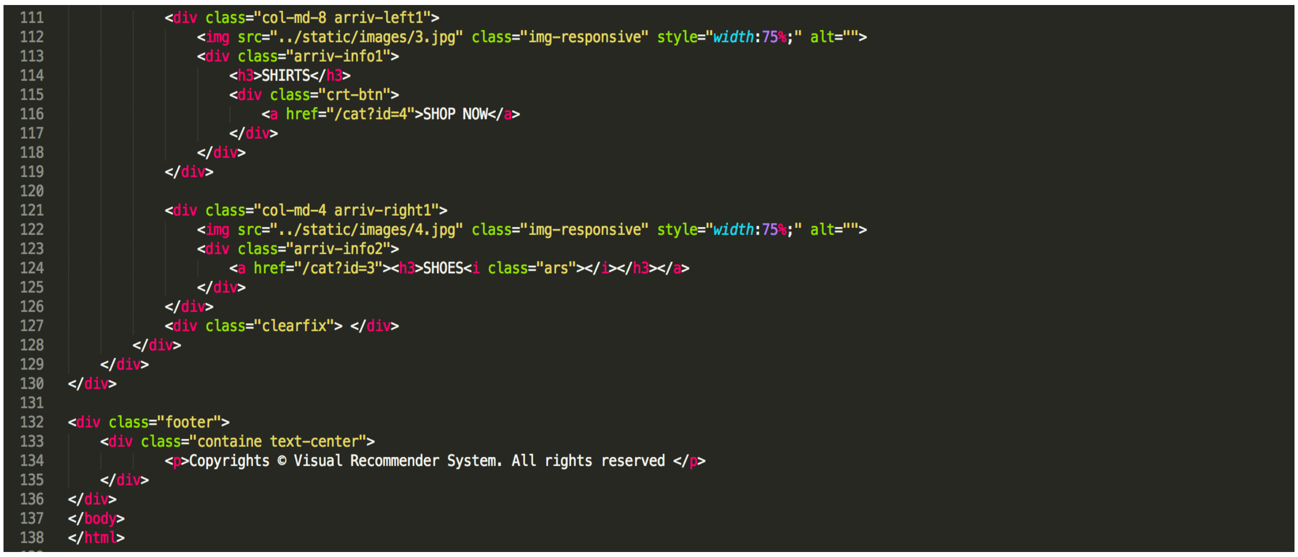
## Index page for outer UI



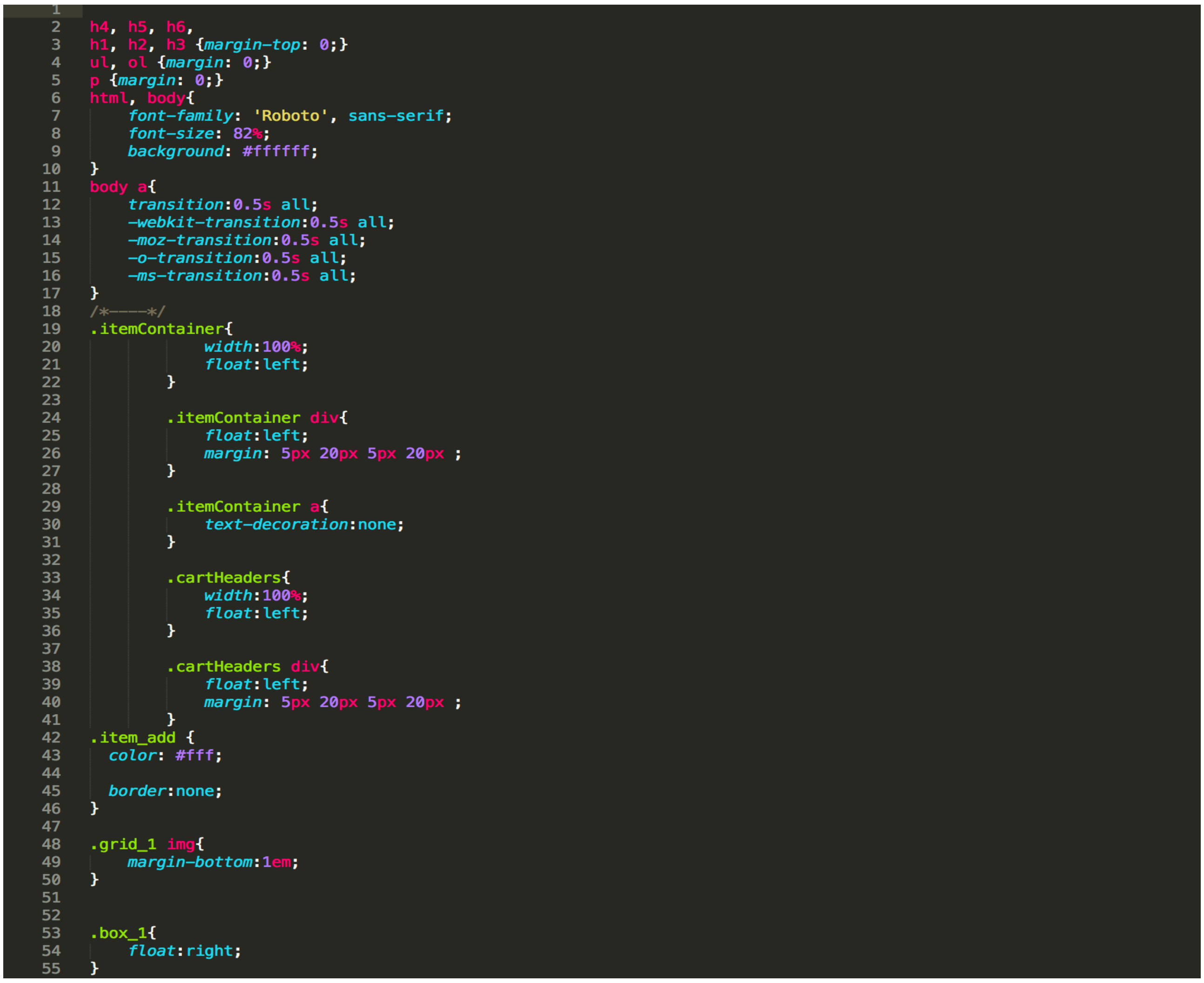
```html
<!DOCTYPE html>
<html>
<head>
    <title>Visual Recommender</title>
    <link href="../static/css/bootstrap.css" rel='stylesheet' type='text/css' />

    <script type='text/javascript' src="../static/js/jquery-1.11.1.min.js"></script>

    <link href="../static/css/style.css" rel='stylesheet' type='text/css' />

    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

    <script type="application/x-javascript">
        addEventListener("load", function() { setTimeout(hideURLbar, 0); }, false);

        function hideURLbar(){
            window.scrollTo(0,1);
        }
    </script>

    <link href='http://fonts.googleapis.com/css?family=Roboto:400,100,300,500,700,900' rel='stylesheet' type='text/css'>
    <link href='http://fonts.googleapis.com/css?family=Playfair+Display:400,700,900' rel='stylesheet' type='text/css'>
    <!-- start menu -->
    <link href="../static/css/megamenu.css" rel="stylesheet" type="text/css" media="all" />
    <script type="text/javascript" src="../static/js/megamenu.js"></script>
    <script>
        $(document).ready(function(){$(".megamenu").megamenu();});
    </script>
    <script src="../static/js/menu_jquery.js"></script>
    <script type="text/javascript">
    function readURL(input) {
            if (input.files && input.files[0]) {
                var reader = new FileReader();

                reader.onload = function (e) {
                    $('#blah')
                        .attr('src', e.target.result);
                };
                reader.readAsDataURL(input.files[0]);
            }
        }
    </script>
</head>

<body>

<div class="header_bg">
<div class="container">
    <div class="header">
    <div class="head-t">
        <div class="logo">
            <center><h1>Visual Recommender System</h1><center>
        </div>
```



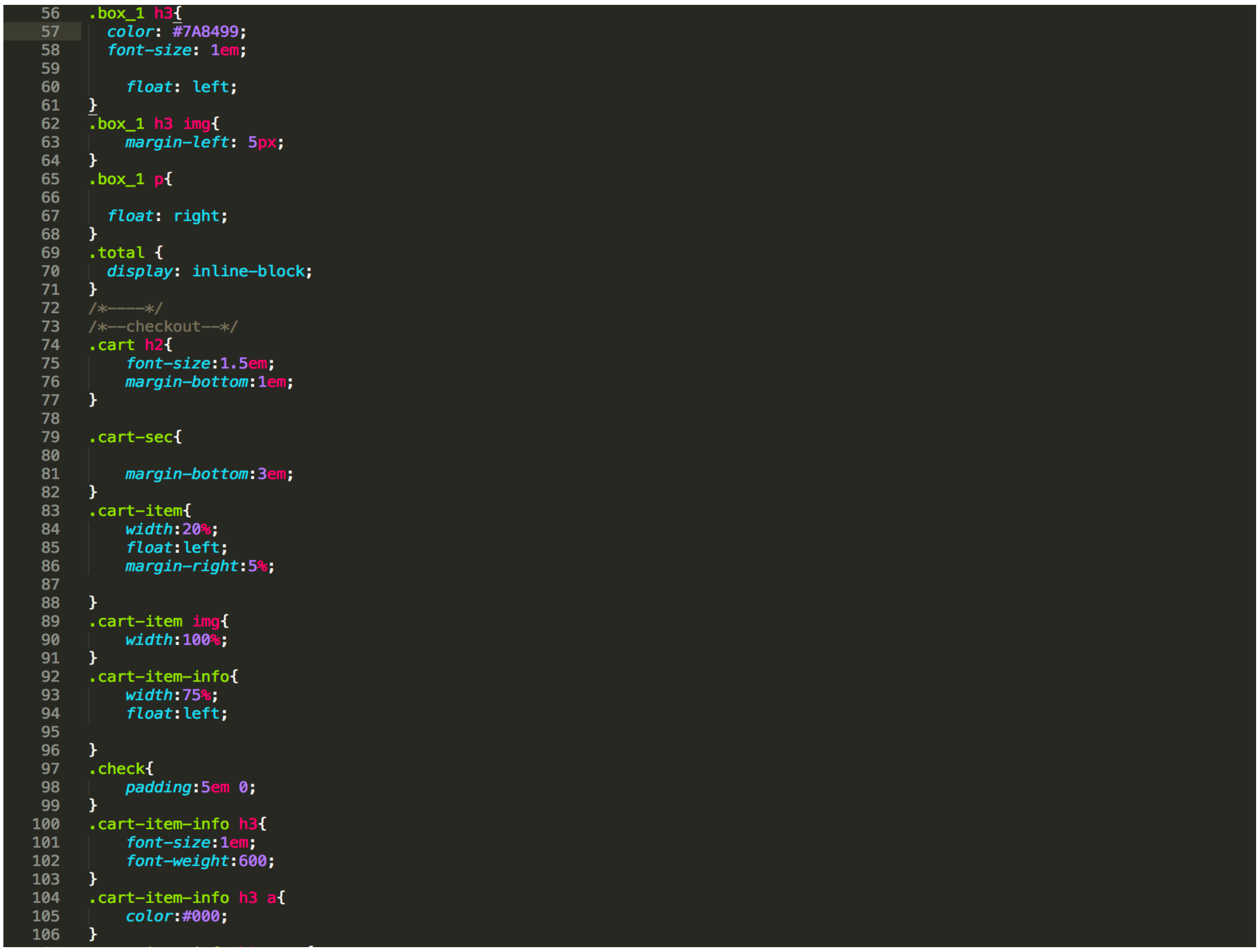
```html

        <!-- start header_right -->
        <div class="header_right">
        <br>
        <div class="clearfix"> </div>
        </div>
        <!--div class="clearfix"> </div-->
    </div>

        <!-- start header menu -->
        <ul class="megamenu skyblue">
            <li class="active grid"><a class="color1" href="/">Home</a></li>
            <li class="grid"><a class="color2" href="/cat?id=1">T-SHIRT</a></li>
            <li><a class="color4" id='4' href="/cat?id=2">JEANS</a></li>
            <li><a class="color5" id='5' href="/cat?id=3">SHOES</a></li>
            <li><a class="color6" id='6' href="/cat?id=4">SHIRTS</a></li>
         </ul>
    </div>
</div>   <!-- ENd of Container -->
</div>
<br>
<div class="arriv">
    <div class="container">
        <div class="arriv-las">
            <div class="col-md-4 arriv-left2">
                <div>
        IMAGE UPLOAD
        <input type='file' onchange="readURL(this);" />
        <div>
       <img id="blah" src="http://placehold.it/180" alt="your image" style="display:block;width:100%;max-width:180px;padding:10px;background:#2d2d2d;" />

        <a href="api?id=105&dirid=1">SEE SIMILAR</a>
    </div>

               </div>
            </div>
            <div class="col-md-4 arriv-middle">
                <img src="../static/images/6.jpg" class="img-responsive" style="width:75%;" alt="">
                <div class="arriv-info3">
                    <h3>T-SHIRT</h3>

                    <div class="crt-btn">
                        <a href="/cat?id=1">SHOP NOW</a>
                    </div>
                </div>
            </div>
            <div class="col-md-4 arriv-right2">
                <img src="../static/images/7.jpg" class="img-responsive" style="width:75%;" alt="">
                <div class="arriv-info2">
                    <a href="/cat?id=2"><h3>JEANS<i class="ars"></i></h3></a>
                </div>
            </div>
            <div class="clearfix"> </div>
        </div>

        <div class="arriv-bottm">
```



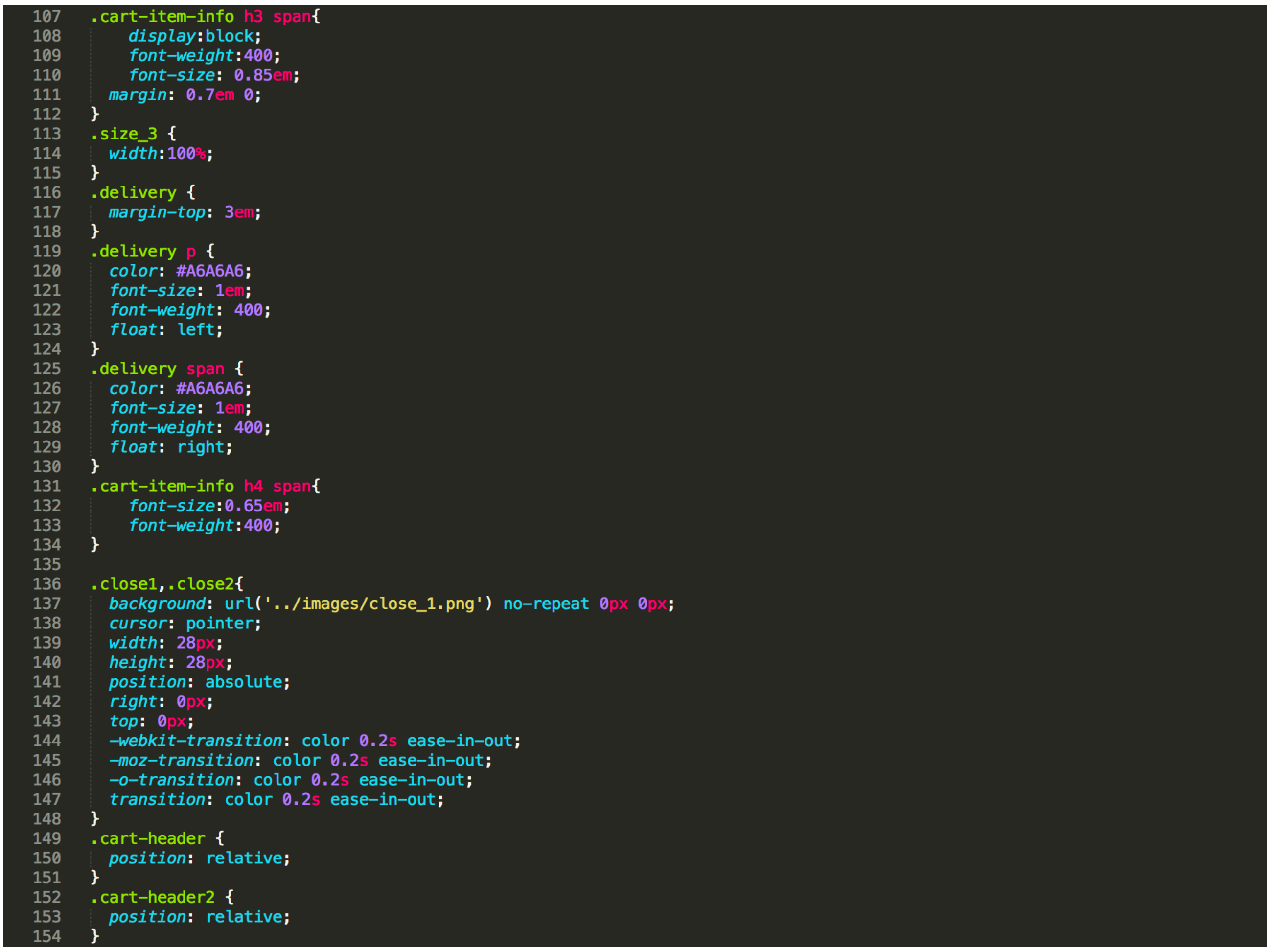
```html
            <div class="col-md-8 arriv-left1">
                <img src="../static/images/3.jpg" class="img-responsive" style="width:75%;" alt="">
                <div class="arriv-info1">
                    <h3>SHIRTS</h3>
                    <div class="crt-btn">
                        <a href="/cat?id=4">SHOP NOW</a>
                    </div>
                </div>
            </div>

            <div class="col-md-4 arriv-right1">
                <img src="../static/images/4.jpg" class="img-responsive" style="width:75%;" alt="">
                <div class="arriv-info2">
                    <a href="/cat?id=3"><h3>SHOES<i class="ars"></i></h3></a>
                </div>
            </div>
            <div class="clearfix"> </div>
        </div>
    </div>
</div>

<div class="footer">
    <div class="containe text-center">
            <p>Copyrights © Visual Recommender System. All rights reserved </p>
    </div>
</div>
</body>
</html>
```



```css

h4, h5, h6,
h1, h2, h3 {margin-top: 0;}
ul, ol {margin: 0;}
p {margin: 0;}
html, body{
    font-family: 'Roboto', sans-serif;
    font-size: 82%;
    background: #ffffff;
}
body a{
    transition:0.5s all;
    -webkit-transition:0.5s all;
    -moz-transition:0.5s all;
    -o-transition:0.5s all;
    -ms-transition:0.5s all;
}
/*----*/
.itemContainer{
            width:100%;
            float:left;
        }

        .itemContainer div{
            float:left;
            margin: 5px 20px 5px 20px ;
        }

        .itemContainer a{
            text-decoration:none;
        }

        .cartHeaders{
            width:100%;
            float:left;
        }

        .cartHeaders div{
            float:left;
            margin: 5px 20px 5px 20px ;
        }
.item_add {
  color: #fff;

  border:none;
}

.grid_1 img{
    margin-bottom:1em;
}


.box_1{
    float:right;
}
```



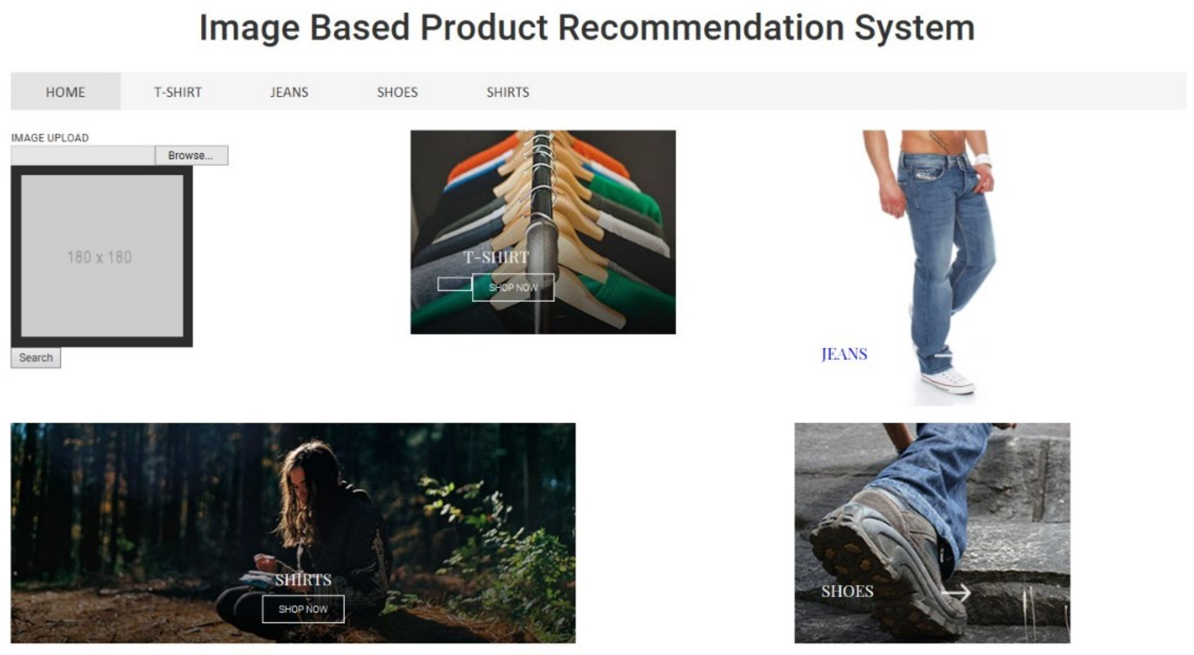
```
.box_1 h3{
  color: #7A8499;
  font-size: 1em;

    float: left;
}
.box_1 h3 img{
    margin-left: 5px;
}
.box_1 p{

  float: right;
}
.total {
  display: inline-block;
}
/*----*/
/*--checkout--*/
.cart h2{
    font-size:1.5em;
    margin-bottom:1em;
}

.cart-sec{

    margin-bottom:3em;
}
.cart-item{
    width:20%;
    float:left;
    margin-right:5%;

}
.cart-item img{
    width:100%;
}
.cart-item-info{
    width:75%;
    float:left;

}
.check{
    padding:5em 0;
}
.cart-item-info h3{
    font-size:1em;
    font-weight:600;
}
.cart-item-info h3 a{
    color:#000;
}
```

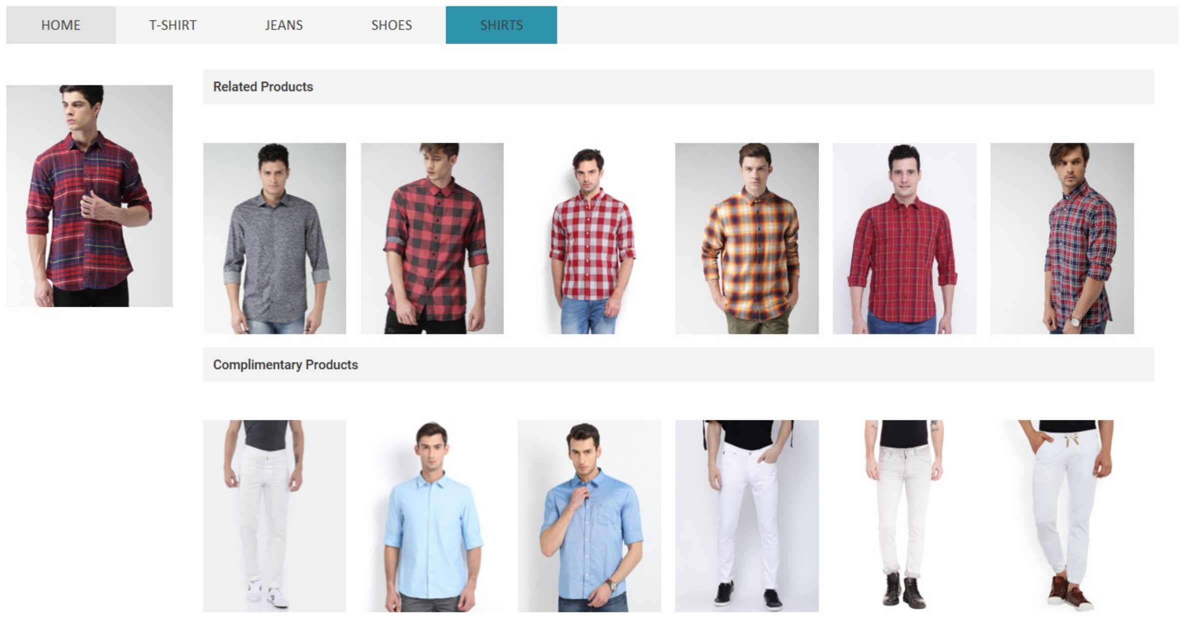


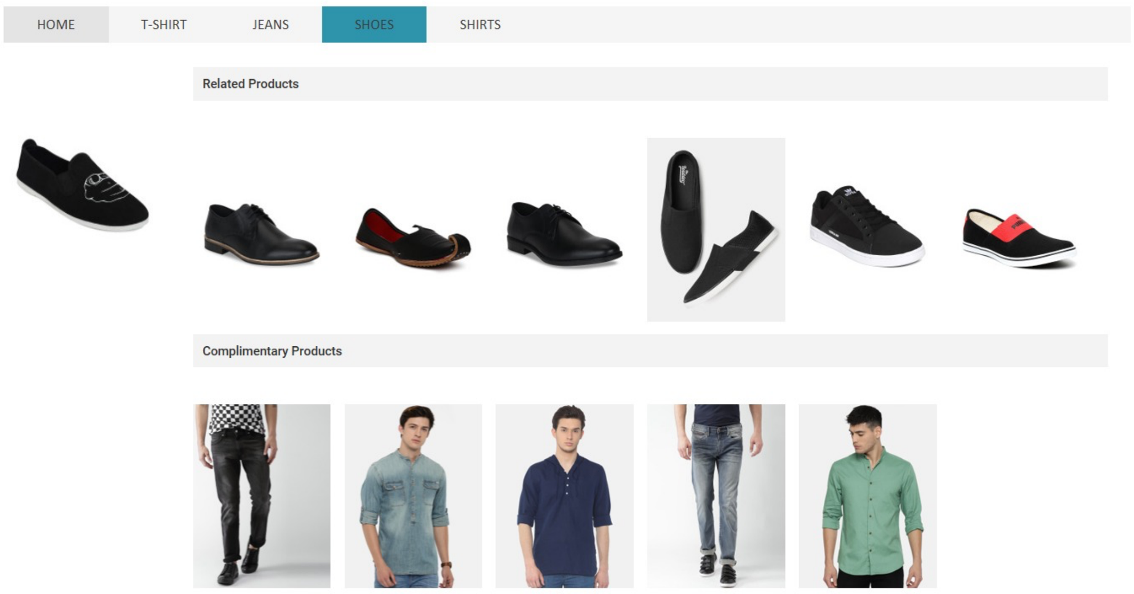
```css
.cart-item-info h3 span{
    display:block;
    font-weight:400;
    font-size: 0.85em;
  margin: 0.7em 0;
}
.size_3 {
  width:100%;
}
.delivery {
  margin-top: 3em;
}
.delivery p {
  color: #A6A6A6;
  font-size: 1em;
  font-weight: 400;
  float: left;
}
.delivery span {
  color: #A6A6A6;
  font-size: 1em;
  font-weight: 400;
  float: right;
}
.cart-item-info h4 span{
    font-size:0.65em;
    font-weight:400;
}

.close1,.close2{
  background: url('../images/close_1.png') no-repeat 0px 0px;
  cursor: pointer;
  width: 28px;
  height: 28px;
  position: absolute;
  right: 0px;
  top: 0px;
  -webkit-transition: color 0.2s ease-in-out;
  -moz-transition: color 0.2s ease-in-out;
  -o-transition: color 0.2s ease-in-out;
  transition: color 0.2s ease-in-out;
}
.cart-header {
  position: relative;
}
.cart-header2 {
  position: relative;
}
```

## Outcome

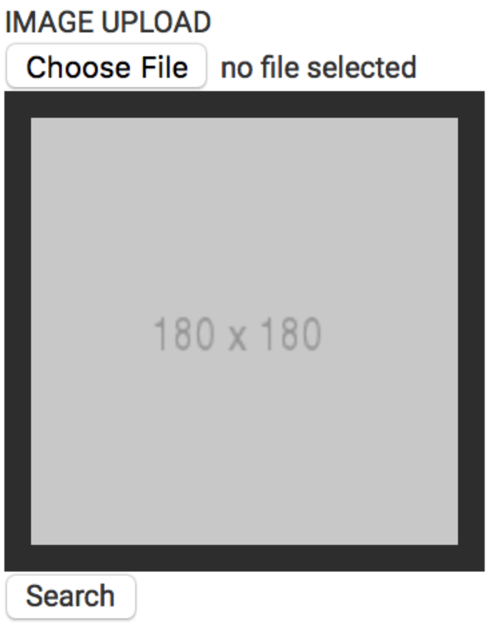
### Known Product Scenario

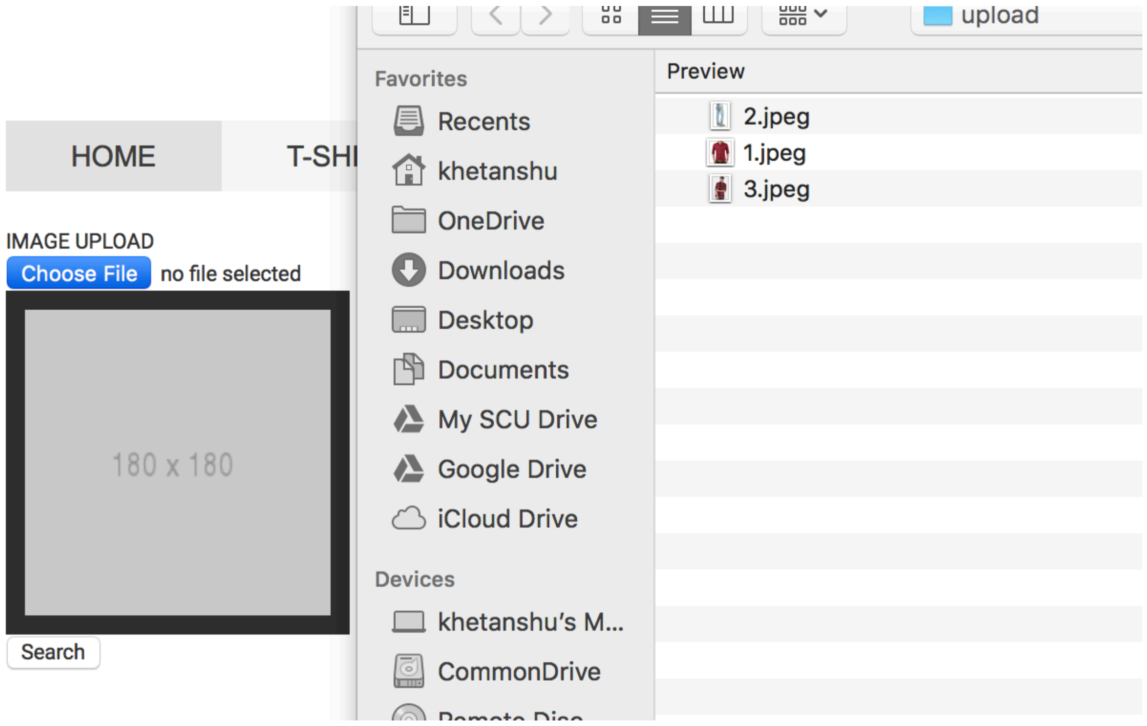


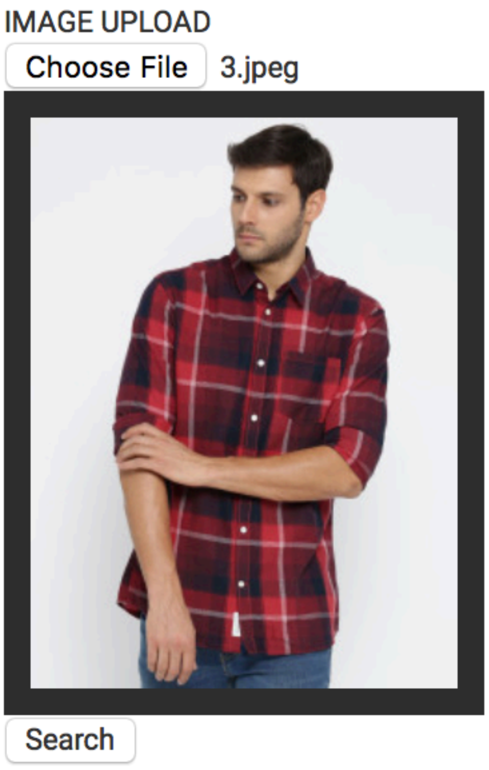


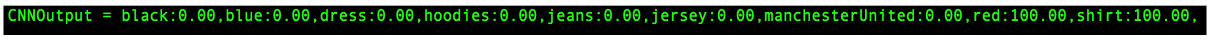

HOME
T-SHIRT
JEANS
SHOES
SHIRTS
Related Products
Complimentary Products

## Unknown Product Scenario

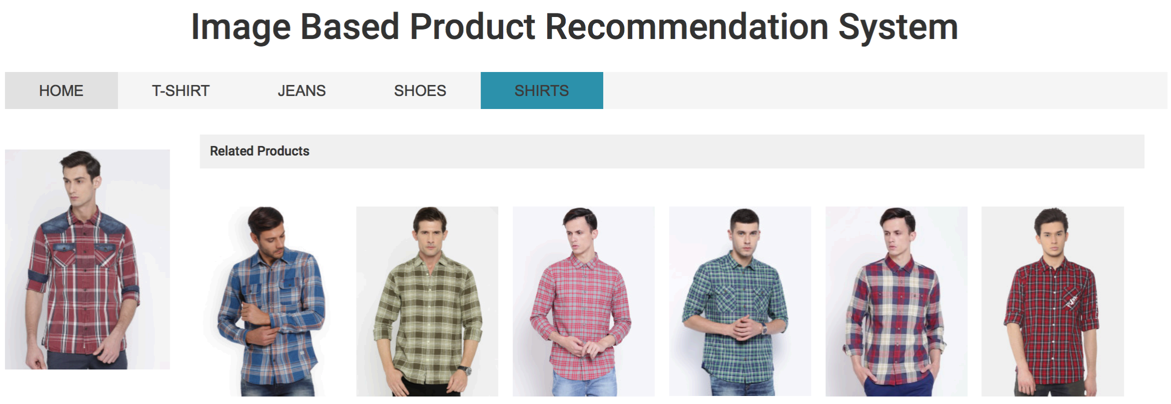








```
CNNOutput = black:0.00,blue:0.00,dress:0.00,hoodies:0.00,jeans:0.00,jersey:0.00,manchesterUnited:0.00,red:100.00,shirt:100.00,
```



# Conclusions and Recommendation

## Summary and Conclusions

In this project we build a smart shopping recommender for image search. We tried out different neural network models for image classification and different ways to quantify the similarity between two images. We are able to achieve a classification accuracy of 0.98 and recommend products with similarity score higher than 0.98.

## Recommendations for Future Work

There is over-fitting issue in our model, which can be one of the things to do in future work. As shown in the Dataset and Features section, though we have a huge data set, due to the limitation on time and machine memory, we only used 10,000 out of 3.5 million images. In the next step, we can try to train our model on a larger amount of data using batches. This can potentially increase the accuracy of the model.

# Bibliography

1. T. Deselaers and V. Ferrari. Visual and semantic similar- ity in imagenet. In Computer Vision and Pattern Recogni- tion (CVPR), 2011 IEEE Conference on, pages 1777–1784. IEEE, 2011.
2. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.
3. M.Tan,S.-P.Yuan,andY.-X.Su.Alearning-basedapproach to text image retrieval: using cnn features and improved sim- ilarity metrics. arXiv preprint arXiv:1703.08013, 2017.
4. Rogers and Nicewander (1988). "Thirteen Ways to Look at the Correlation Coefficient" (PDF). The American Statistician.
5. Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In Proc. NIPS.
6. Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.Com Recommenda- tions: Item-to-Item Collaborative Filtering. IEEE Internet Computing 7, 1 (Jan. 2003), 76–80.
7. S. Liu, J. Feng, C. Domokos, H. Xu, J. Huang, Z. Hu, and S. Yan. 2014. Fashion Parsing With Weak Color-Category Labels. IEEE Transactions on Multimedia 16, 1 (Jan 2014), 253–265.
8. J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu. Learning fine-grained image similarity with deep ranking. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1386–1393, 2014.