Intelligence Beyond Fashion

Group Name: Group 1

Group Members:

First name	Last Name	Student number
Nikeshh	Vijayabaskaran	C0849544
Jebin	George	C0850509
Deeksha	Naikap	C0835440
Shravan Kumar Reddy	Gunthala	C0833124
Raghavendra Reddy	Putta Hanumantha Reddy	C0851724

Submission date: 22nd April 2023

Contents

1. Abstract

Intelligence Beyond Fashion is a unique project that aims to provide users with a seamless experience when it comes to online shopping for fashion products. The project focuses on addressing the problem of impulse buying, which is a common issue faced by many shoppers. With so many products available across various platforms, it can be challenging for users to make an informed decision before purchasing a product. This project aims to provide a solution to this problem by comparing product details like product titles and images across multiple different products from different platforms.

One of the key features of Intelligence Beyond Fashion is the availability of multiple inbound channels for users to interact with. Users can access the project through a website, app, and chrome extension. These multiple channels make it convenient for users to access the project from anywhere and at any time.

The project leverages the power of artificial intelligence to analyze product details, images, and other information to provide users with an accurate comparison of products. This comparison helps users to make an informed decision before making a purchase. The project also takes into account the user's preferences and history to provide personalized recommendations that cater to their individual needs.

Intelligence Beyond Fashion is a project that aims to revolutionize the way people shop online for fashion products. With its unique features and advanced technology, the project provides a convenient and personalized shopping experience for users, while also addressing the issue of impulse buying.

2. Introduction

This project mainly focuses on two main components:

1. TrendAnalysis – To find the most relevant trends using topic modelling
2. TrendMatch – To match the given product title and images across a set of different products from different merchant platforms. For simplifying, we have scraped all the data from amazon. This is done using the help of CNN model

Below flowcharts gives a high-level workflow of the project

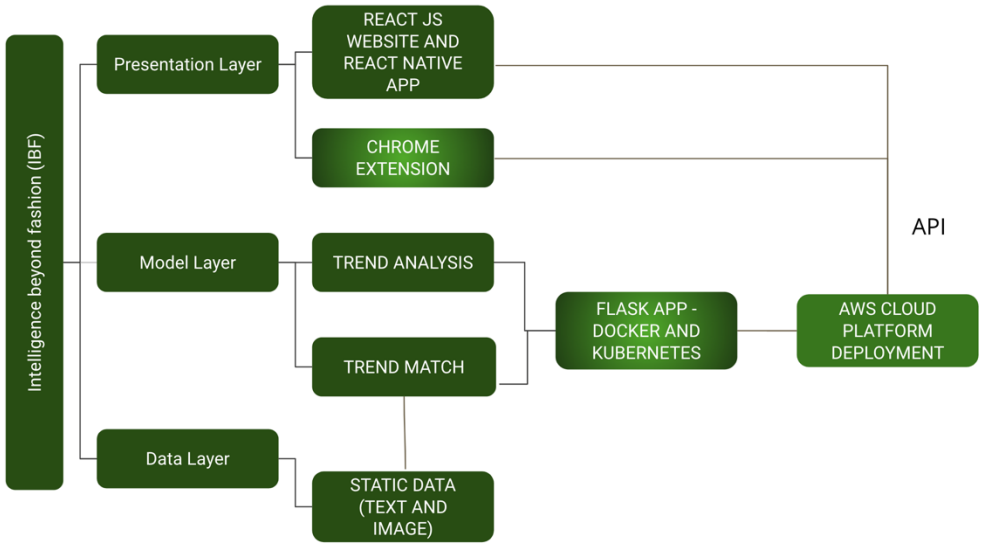


Figure 1 - Project flow

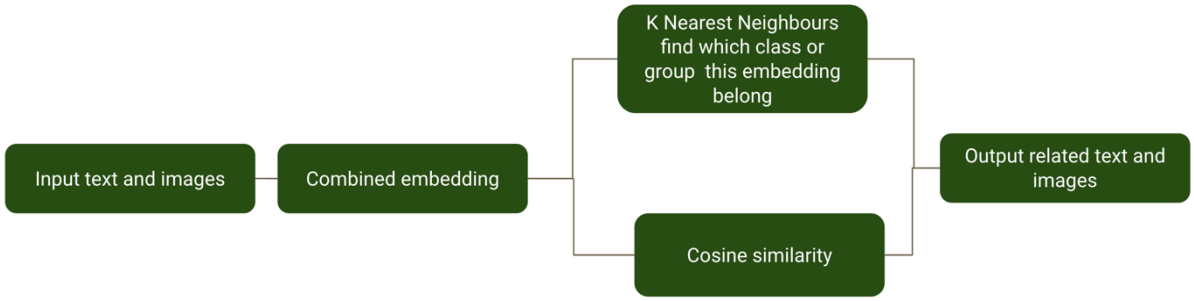


Figure 2 Input data flow and prediction

There are three different layers namely:

1. Presentation layer which forms the components that is visually available to the end user or the stakeholder

   Three different types of inbound channels are implemented:

   a. Website developed using ReactJS library

   b. Android and IOS – Cross-platform application developed using React Native framework

c. Google chrome extension

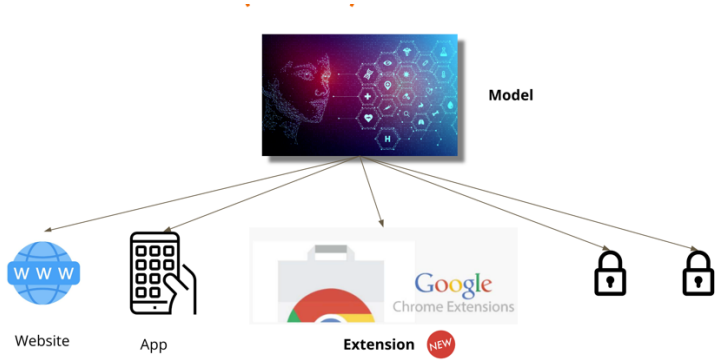


Figure 3 - Inbound channels

2. Model layer which holds the machine learning model to process the data

    a. TrendAnalysis – To find the most relevant trends using topic modelling

    b. TrendMatch – To match the given product title and images across a set of different products from different merchant platforms. For simplifying, we have scraped all the data from amazon. Product matching is done using the help of CNN model

    The complete model is exposed as API (Application Programming Interface) using flask framework. Furthermore, the application is dockerized and deployed in AWS cloud – AWS Elastic Beanstalk. EKS is used as a supplementary to handle the data load

3. Data layer, which holds the data provided to the model

    Data currently scraped from Amazon is directly loaded as input to the model.

The input data, both text and image, is converted into embeddings. These embeddings are compared against the embeddings of products which is already generated. The comparison is made using the following:

1. KMeans and KNN to find the relevant products
2. Cosine values to find the relevant products

3. Methods

Sections ***3.1. to 3.10*** covers the TrendMatch application

3.1. Data Collection

The data collection process can be performed using various tools and technologies, and one such tool is Selenium, a web automation tool. We have used Selenium and Beautiful Soup for scraping product images from Amazon. The data collection process used the following Python libraries:

Selenium - A web automation tool to automate web browsers' interactions with web pages.
Beautiful Soup – It is a library used to extract data out of web files such as XML and HTML.
Requests - A Python library used for sending HTTP requests to web pages and getting responses.

The Python code was written to scrape images of a particular product using Amazon as an example. The code uses an infinite scroll URL, which means the code keeps scrolling down to load more images until a predefined limit is reached. We then used Selenium to open the Chrome browser and load the Amazon page. And scraped the images of the product by looping through the HTML tags and finding the image tags to

extract the image source link. The downloaded images are saved in a folder along with the product's title and price. Finally, the scraped metadata is saved to a CSV file, making it easier to analyze and use in further research or analysis.

It can be noted that in most of the ecommerce websites, the title of the website is enclosed in <h1></h1> tag and image is enclosed with <img src="image url" alt="alternative text">.
This is specifically done for SERP purposes and hence we used the scraping tool to scrape this particular data.

The dataset thus obtained has ***43 different categories and 10921 image data***. The data collection process using Selenium and Beautiful Soup for scraping product images from Amazon is a simple and effective way to collect data.

The following table explains about the different types of major labels available in our collected data arranged in ascending alphabetical order.

ID	Category
1	Backpacks
2	Baseball hats
3	Beanies
4	Bowler Hat
5	Bracelets
6	Bucket hats
7	Cargo Shorts
8	Chronograph watches
9	Denim Jackets
10	Denim Shorts
11	Dress Watches
12	Duffel bags
13	Earrings
14	Formal Shoes
15	Graphic hoodies
16	Leather Jackets
17	Men Formal Shoes
18	Men Sandals
19	Mens Black Shirt
20	Mens Blue Shirt
21	Mens Brown Shirt
22	Mens Grey Shirt
23	Mens White Shirt
24	Messenger bags
25	Necklaces
26	Puffer Jackets
27	Pullover hoodies
28	Rings
29	Running Shoes
30	Shoulder bags
31	Smart Watches
32	Sneakers
33	Sports Shorts

34	Tote bags
35	Trucker Hats
36	Women Black Tops
37	Women Blue Tops
38	Women Formal Shoes
39	Women Pink Tops
40	Women Red Tops
41	Women Sandals
42	Women White Tops
43	Zip-up hoodies

The below bar graph gives an idea of the number of images contained in each class.

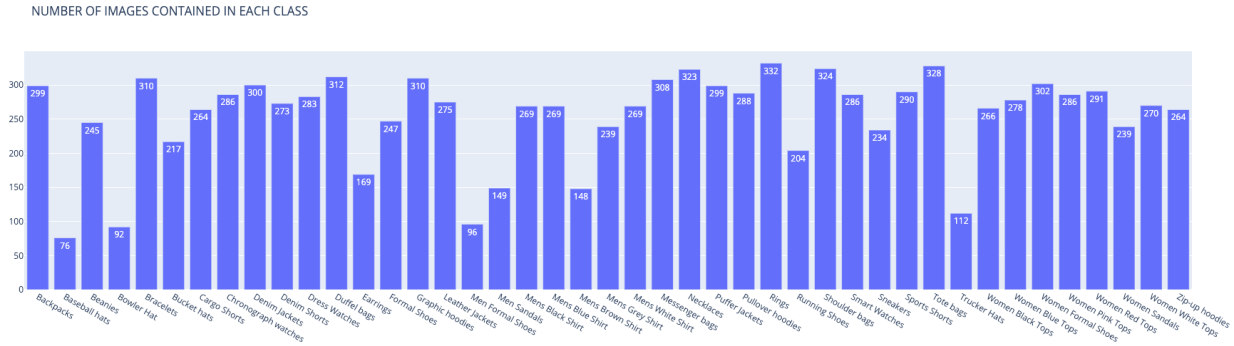


Figure 4 - Number of images contained in each class

All 43 categories are considered as different classes. Below is the list of classes in the dataset.

3.2. Data Preprocessing

In machine learning, image data preprocessing refers to a series of actions taken on raw image data to make it suitable for ML model use. The primary aim of image preprocessing is to enhance image quality, eliminate noise or unwanted elements that might disrupt analysis, and extract useful features that can be utilized for image classification, object detection, segmentation, or other tasks related to images. Some common techniques used in image preprocessing include:

- ***Resizing and scaling:*** This involves adjusting the size of the image to fit a specific model or to reduce computational complexity.
- ***Normalization:*** This involves rescaling the pixel values of an image to a common scale in order to reduce the effects of lighting and contrast variations.
- ***Image enhancement:*** This includes techniques such as contrast stretching, histogram equalization, and edge enhancement, which aim to improve the visual quality of an image.
- ***Noise reduction:*** This involves filtering techniques such as median filtering, Gaussian smoothing, and bilateral filtering, which aim to remove unwanted noise from an image.

- ***Feature extraction:*** This involves identifying and extracting relevant features from the image that can be used for classification or other machine learning tasks.

- **Data augmentation:** Several transformations are applied to the original image data, to increase the number of data samplings. Flipping, rotation, shearing, cropping, and other transformations are applied to the initial dataset and new samples are created. This way the diversity of the dataset is enhanced, and the model generalization capability is improved.

- ***Grayscaling:*** Converting a color image into a grayscale image is a widely used method in image preprocessing. The process involves creating a 2D matrix of pixel values that represents the brightness or intensity of each pixel in the original image. This grayscale image is essentially a black and white version of the original colour image.

Image preprocessing is an important step in many machine learning tasks involving image data, as the accuracy and efficiency of the model can be improved in this way.

The following operations are done to preprocess the image and prepare the image data for further processing.

Shifting

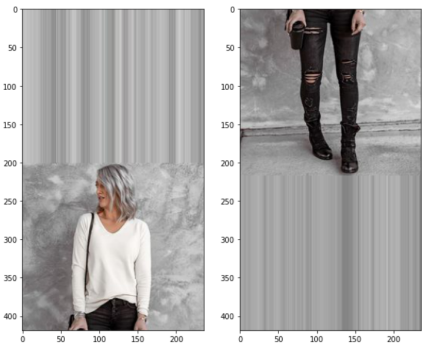


Figure 5 - Shifting

Flipping

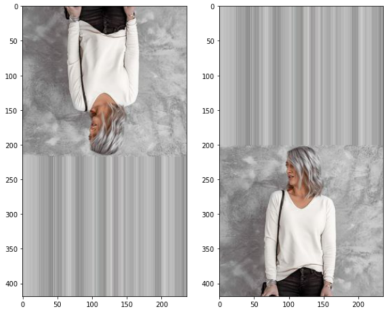


Figure 6 - Flipping

Rotation

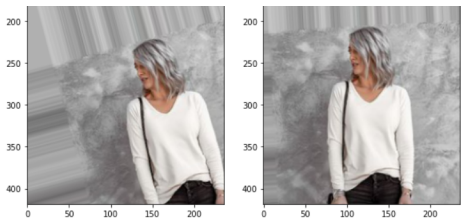


Figure 7 - Rotation

Grayscale conversion

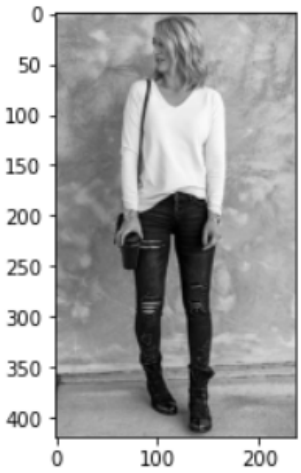


Figure 8 - Grayscale conversion

3.3. Train-test split

Dividing the data into testing and training sets is an essential process in machine learning. The main objective of this step is to assess the model's performance on data on which it is not trained before. By using a distinct dataset to evaluate the model's performance, we can obtain an approximation of when exposed to new, unobserved data in real-world scenarios, how the performance of the model is.

Using the training dataset, the model is trained, and the test dataset is used for evaluating its performance. If the model performs well on the test set, it suggests that its performance will be good on the new dataset. In contrast, if the model performs poorly on the test set, it might be overfitting, and we may need to adjust our model or training process. When the model is too complex and the training data is fitted too closely, overfitting happens. To avoid this, the division of the dataset into train and test is helpful. By having a separate test set, we can detect whether the model is overfitting or not, as the test set's performance will decrease in case there is overfitting in the model.

To divide the dataset into testing and training, the function train_test_split is used. ***20% of the input data is reserved for testing, while 80% is used for training the model.***

```
X_train.shape
```

```
(8539, 120, 120, 3)
```

Figure 9 - X train data shape

```
X_test.shape
```

```
(2135, 120, 120, 3)
```

Figure 10 - Test set shape

After the split the train data has 8539 images, whereas the test has 2135 image data.

Further, the train is split, and 10% is reserved as the validation data.

3.4. Model build

We have implemented four different types of models, namely

1. **Autoencoder decoder image model (Type 1)**
2. **Autoencoder decoder image model (Type 2)**
3. **CNN image model**
4. **Efficient B5 pre-trained model (Only used for comparison)**

However, it should be noted that the base of the autoencoder decoder image model is a Convolutional Neural Networks (CNN) model.

CNN is an artificial neural network type that specializes in processing image data. Their primary function is to classify images. They are made up of multiple layers, such as pooling layers, convolutional layers, dense layers, and others. The features are extracted from the input image using filters in convolutional layers. The dimensionality of the convolutional layer's output is reduced at pooling layers. Finally, based on the extracted features, the classification of the image is done at fully connected layers. The ability of CNNs to ***learn features automatically without the need for manual feature extraction*** has made them increasingly popular in recent years. They have been beneficial in various fields, such as computer vision, self-driving cars, and medical imaging. This has led to significant improvements in image recognition and classification tasks.

Here is the pictorial representation of a Convolutional Neural Network Model

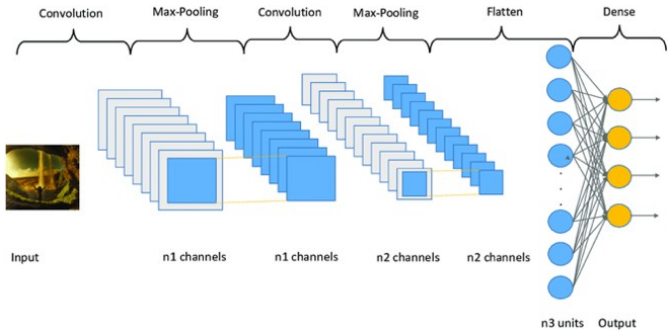


Figure 11 - CNN

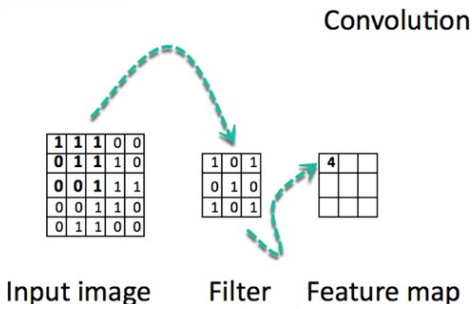


Figure 12 - Filters for CNN

3.4.1. Autoencoder decoder image model (Type 1)

These type of neural network models are constructed based on the approach of compressing and reconstructing the data given.
This helps the model to understand the data better and handle heavy loads of data.
This model has the following structure:

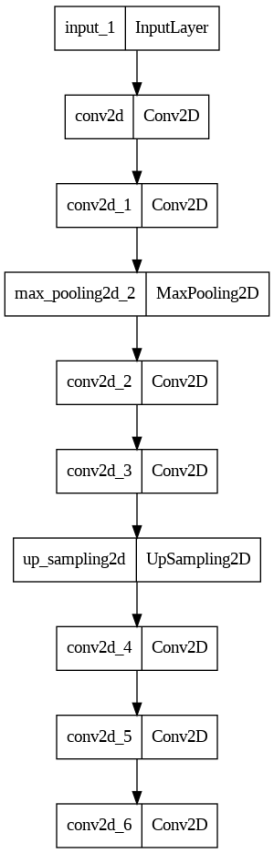


Figure 13 - Autoencoder decoder model structure

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 conv2d (Conv2D)             (None, 224, 224, 224)     6272

 conv2d_1 (Conv2D)           (None, 224, 224, 128)     258176

 max_pooling2d_2 (MaxPooling  (None, 112, 112, 128)    0
 2D)

 conv2d_2 (Conv2D)           (None, 112, 112, 64)      73792

 conv2d_3 (Conv2D)           (None, 112, 112, 64)      36928

 up_sampling2d (UpSampling2D  (None, 224, 224, 64)     0
 )

 conv2d_4 (Conv2D)           (None, 224, 224, 128)     73856

 conv2d_5 (Conv2D)           (None, 224, 224, 224)     258272

 conv2d_6 (Conv2D)           (None, 224, 224, 3)       6051

=================================================================
Total params: 713,347
Trainable params: 713,347
Non-trainable params: 0
```

Figure 14 - Model structure

This model has two main section,
First section which is the encoder or encoding architecture which take the input image and starts compressing the images.
In the second section which is the decoder or decoding architecture which takes the compressed images and reconstructs the images

3.4.2. Autoencoder decoder image model (Type 2)

This is also a type of autoencoder decoder model but the number of convolution layers in both encoder and decoder is increased.
This model has the following structure:

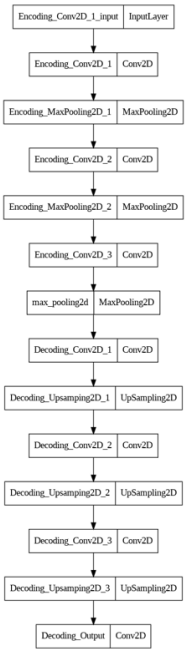


Figure 15 - Model structure

```
Model: "Convolutional_AutoEncoder_Decoder_Second_Image_Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Encoding_Conv2D_1 (Conv2D)   (None, 224, 224, 224)     6272

Encoding_MaxPooling2D_1 (Ma  (None, 112, 112, 224)     0
xPooling2D)

Encoding_Conv2D_2 (Conv2D)   (None, 112, 112, 128)     258176

Encoding_MaxPooling2D_2 (Ma  (None, 56, 56, 128)       0
xPooling2D)

Encoding_Conv2D_3 (Conv2D)   (None, 56, 56, 64)        73792

max_pooling2d (MaxPooling2D  (None, 28, 28, 64)        0
)

Decoding_Conv2D_1 (Conv2D)   (None, 28, 28, 64)        36928

Decoding_Upsamping2D_1 (UpS  (None, 56, 56, 64)        0
ampling2D)

Decoding_Conv2D_2 (Conv2D)   (None, 56, 56, 128)       73856

Decoding_Upsamping2D_2 (UpS  (None, 112, 112, 128)     0
ampling2D)

Decoding_Conv2D_3 (Conv2D)   (None, 112, 112, 224)     258272

Decoding_Upsamping2D_3 (UpS  (None, 224, 224, 224)     0
ampling2D)

Decoding_Output (Conv2D)     (None, 224, 224, 3)       6051

=================================================================
Total params: 713,347
Trainable params: 713,347
Non-trainable params: 0
_________________________________________________________________
```

Figure 16 - Model structure

3.4.3. CNN image model

Using the Keras library, we have developed the Convolutional Neural Network model. Here is the model architecture.:

- This model is a sequential neural network architecture built using Keras library in Python. It

is designed for image classification tasks and takes 120x120 pixel RGB images as input.

- The model consists of several layers, including Conv2D, MaxPooling2D, Flatten, Dense and Dropout layers.
- 1st Conv2D Layer: This CNN layer has 64 filters each of size 3x3 with a stride of 1 and valid padding. It takes an input shape of 120x120x3 (width x height x channels) and applies the ReLU activation function.
- MaxPooling2D Layer 1: This layer performs max pooling operation, and the size of the pool is 2x2. It reduces the first convolutional layer's output's spatial dimensions.
- 2nd Conv2D Layer: It has 128 filters each of size 3x3 with 2 strides and the same padding. To the output of the layer, it applies the activation function ReLU.
- 2nd MaxPooling2D Layer: The max pooling operation is performed with a 2x2 pool size. It reduces the second convolutional layer's output's spatial dimensions.
- 3rd Conv2D Layer: It has 64 filters each of size 3x3 with 2 strides and same padding. To the output of the layer, it applies the activation function ReLU.
- 3rd MaxPooling2D Layer: The max pooling operation is performed with a 2x2 pool size. It reduces the third convolutional layer's output's spatial dimensions.
- Flatten Layer: It compresses the previous layer's output into an array of 1 dimension. And it is then provided to the dense layer as input.
- 1st Dense Layer: It has 128 neurons and applies the ReLU activation function.
- Dropout Layer 1: To prevent overfitting, 25% of the input units are dropped out randomly in this layer during training.
- 2nd Dense Layer: This layer has 256 neurons and applies the ReLU activation function.
- Dropout Layer 2: This layer drops out 50% of the input units randomly during training to prevent overfitting.
- Dense Layer 3: This layer has 256 neurons and applies the ReLU activation function.
- Dropout Layer 3: To prevent overfitting, 25% of the input units are dropped out randomly in this layer during training.
- 4th Dense Layer: This layer has 128 neurons and applies the ReLU activation function.
- Dropout Layer 4: This layer drops out 10% of the input units randomly during training to prevent overfitting.
- Dense Layer 5: This is the output layer with 42 units and the activation function of softmax, which outputs the input image probabilities that belongs to all 42 possible classes.

This architecture is designed to learn and extract the input image data features through the pooling and convolutional layers, which are then flattened and fed into fully connected layers for classification. The use of dropout layers helps to prevent overfitting.

```
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_21 (Conv2D)           (None, 118, 118, 64)      1792

max_pooling2d_21 (MaxPoolin  (None, 59, 59, 64)        0
g2D)

conv2d_22 (Conv2D)           (None, 30, 30, 128)       73856

max_pooling2d_22 (MaxPoolin  (None, 15, 15, 128)       0
g2D)

conv2d_23 (Conv2D)           (None, 8, 8, 64)          73792

max_pooling2d_23 (MaxPoolin  (None, 4, 4, 64)          0
g2D)

flatten_7 (Flatten)          (None, 1024)              0

dense_35 (Dense)             (None, 128)               131200

dropout_28 (Dropout)         (None, 128)               0

dense_36 (Dense)             (None, 256)               33024

dropout_29 (Dropout)         (None, 256)               0

dense_37 (Dense)             (None, 256)               65792

dropout_30 (Dropout)         (None, 256)               0

dense_38 (Dense)             (None, 128)               32896

dropout_31 (Dropout)         (None, 128)               0

dense_39 (Dense)             (None, 42)                5418

=================================================================
Total params: 417,770
Trainable params: 417,770
Non-trainable params: 0
_________________________________________________________________
```

Figure 17 - Model structure

3.4.4. Efficient B5 Pretrained model

We chose Efficient B5 image model which is a pretrained model for comparison.
This model was further trained on learning transfer using the current image dataset that we scraped.

3.5. Train the Model

Different models were trained with different kinds of parameters, but however for comparison, at least one set of unique parameters and metrics was used. For all the below models, early stopper of patience level 10 and monitor for the validation loss was used.

3.5.1. Autoencoder decoder image model (Type 1)

This model was trained with the following parameters:

1. Optimizer: Adam
2. Loss: mse – mean square error
3. metrics - accuracy

3.5.2. Autoencoder decoder image model (Type 2)

In this model, Hyperparameter tuning was done for this model by using the following parameters:

1. Loss: mse – mean square error
2. Metrics – accuracy
3. Optimizer as given below

ID	Optimizer	Learning rates
1	Adagrad	0.01
2	Adagrad	0.001
3	Adagrad	0.0001

4	Adagrad	0.00001

ID	Optimizer	Learning rates
1	Adam	0.01
2	Adam	0.001
3	Adam	0.0001
4	Adam	0.00001

ID	Optimizer	Learning rates
1	Rmsprop	0.01
2	Rmsprop	0.001
3	Rmsprop	0.0001
4	Rmsprop	0.00001

3.5.3. CNN image model

We compiled the model using the 'adam' optimizer. The optimizer is used to update the model parameters during training to minimize the loss function. The loss function used in this case is 'categorical_crossentropy', which is commonly used for multiclass classification problems. In order to monitor the model's performance during training, the 'accuracy' object is also specified.

The training of the model on the training dataset for 80 epochs with a batch size of 128, and its performance is monitored using the 'accuracy' metric and a separate validation dataset.

```
model.fit(X_train,Y_train,epochs = 20,batch_size=128,verbose = 1, validation_data=(X_val,y_val))

Epoch 1/20
61/61 [==============================] - 157s 2s/step - loss: 3.7117 - accuracy: 0.0288 - val_loss: 3.6402 - val_accuracy: 0.0340
Epoch 2/20
61/61 [==============================] - 117s 2s/step - loss: 3.5252 - accuracy: 0.0673 - val_loss: 3.2798 - val_accuracy: 0.1311
Epoch 3/20
61/61 [==============================] - 117s 2s/step - loss: 3.1424 - accuracy: 0.1330 - val_loss: 2.7274 - val_accuracy: 0.2037
Epoch 4/20
61/61 [==============================] - 117s 2s/step - loss: 2.7276 - accuracy: 0.1883 - val_loss: 2.3622 - val_accuracy: 0.2658
Epoch 5/20
61/61 [==============================] - 118s 2s/step - loss: 2.4425 - accuracy: 0.2448 - val_loss: 2.1314 - val_accuracy: 0.3162
Epoch 6/20
61/61 [==============================] - 117s 2s/step - loss: 2.2217 - accuracy: 0.2958 - val_loss: 2.0714 - val_accuracy: 0.3724
Epoch 7/20
61/61 [==============================] - 117s 2s/step - loss: 2.1096 - accuracy: 0.3214 - val_loss: 1.8059 - val_accuracy: 0.4368
Epoch 8/20
61/61 [==============================] - 117s 2s/step - loss: 1.9866 - accuracy: 0.3595 - val_loss: 1.7208 - val_accuracy: 0.4426
Epoch 9/20
61/61 [==============================] - 117s 2s/step - loss: 1.8680 - accuracy: 0.3841 - val_loss: 1.7862 - val_accuracy: 0.4438
Epoch 10/20
61/61 [==============================] - 117s 2s/step - loss: 1.7731 - accuracy: 0.4203 - val_loss: 1.6553 - val_accuracy: 0.4836
Epoch 11/20
61/61 [==============================] - 118s 2s/step - loss: 1.6855 - accuracy: 0.4449 - val_loss: 1.6601 - val_accuracy: 0.4813
Epoch 12/20
61/61 [==============================] - 118s 2s/step - loss: 1.6426 - accuracy: 0.4600 - val_loss: 1.6776 - val_accuracy: 0.4801
Epoch 13/20
61/61 [==============================] - 118s 2s/step - loss: 1.6108 - accuracy: 0.4717 - val_loss: 1.5734 - val_accuracy: 0.5141
Epoch 14/20
61/61 [==============================] - 117s 2s/step - loss: 1.4743 - accuracy: 0.5135 - val_loss: 1.6029 - val_accuracy: 0.4965
Epoch 15/20
61/61 [==============================] - 117s 2s/step - loss: 1.4920 - accuracy: 0.5053 - val_loss: 1.5173 - val_accuracy: 0.5328
Epoch 16/20
61/61 [==============================] - 118s 2s/step - loss: 1.4066 - accuracy: 0.5279 - val_loss: 1.4739 - val_accuracy: 0.5375
Epoch 17/20
61/61 [==============================] - 117s 2s/step - loss: 1.3968 - accuracy: 0.5313 - val_loss: 1.5065 - val_accuracy: 0.5480
Epoch 18/20
61/61 [==============================] - 117s 2s/step - loss: 1.3859 - accuracy: 0.5348 - val_loss: 1.5155 - val_accuracy: 0.5340
Epoch 19/20
61/61 [==============================] - 116s 2s/step - loss: 1.3295 - accuracy: 0.5520 - val_loss: 1.4912 - val_accuracy: 0.5621
Epoch 20/20
61/61 [==============================] - 115s 2s/step - loss: 1.2632 - accuracy: 0.5740 - val_loss: 1.4527 - val_accuracy: 0.5492
<keras.callbacks.History at 0x249690dc430>
```

Figure 18 - Model training

3.5.4. Pretrained model

Pretrained model efficient b5 is already trained with the image data. We further trained with our image data and obtain the embeddings. Furthermore, the arc margin or arc soft max layer was generated for the images.

3.6. Model Evaluation

Model evaluation is nothing but, evaluating on a given dataset how good the performance of an ML model is. The model evaluation's aim is to assess the model's pattern learning effectiveness in the data and how well it performs on a new dataset. To assess the machine learning model's performance, various evaluation metrics are applied, and depending upon the nature of the model and the problem statement it varies. Below are some of the commonly used evaluation metrics:

a. Accuracy: The ratio of dataset samples that are classified correctly.
b. Precision: A metric that assesses the positive predictions' dependability by calculating the ratio of true positives to all positive predictions.
c. Recall: From the dataset out of all actual positives, the ratio of true positives. It is a measure to calculate how well the model can identify positive samples.
d. F1 score: It is the arithmetic mean of recall and precision. It is a balanced weight of recall and precision.

To ensure that a machine learning model can generalize to new data, it is crucial to assess its performance on a separate test set instead of the training set. Cross-validation can also be employed to evaluate the model on various subsets of the data.

Evaluation metrics

During the training of a deep learning model, for each epoch, the accuracy and loss of validation and training are stored in the history object. The following keys are used to access these values:

a. 'loss': Training loss for each epoch.
b. 'accuracy': Training accuracy for each epoch.
c. 'val_loss': For each epoch, loss of validation.
d. 'val_accuracy': For each epoch, the accuracy of validation.

These metrics are commonly used in machine learning to assess a model's performance during training and testing.

a. Training loss measures the model's error on the training dataset during one iteration of training, and the aim is to minimize this error as much as possible.
b. Training accuracy measures the percentage of correctly predicted samples in the training dataset, and it indicates how well the training data is fitted in the model.
c. The validation loss determines the model's mistake on the validation set in a single training iteration. The validation dataset is used to measure the performance of the model on new data and refers to a portion of the collected data that is excluded from the training process.
d. Validation accuracy measures the percentage of correctly predicted samples in the validation dataset, and it shows the generalization of the model to new data.

The objective is to reduce the loss of validation and training with increasing accuracy of validation and training, which refers to the fitting of the training data in the model being good and new data being effectively generalized.

3.6.1. Autoencoder decoder image model (Type 1)

We got a train accuracy of 65% which was lower than what we expected.
The loss value and accuracy of the model are as below:

Dataset	Loss	Accuracy	Val_Loss	Val_Accuracy
Train	0.0015	0.6502 – 65%	0.0024	0.8676 – 86%
Test	0.0023	62%	-	-

Prediction

Below is the decoded image for a given image using this model

```
sample_image = train_data[7]
sample_image = np.expand_dims(sample_image,axis=0)
image = autoencoder_decoder_first_image_model.predict(sample_image)
plot_custom(sample_image[0,:,:,:],'','',1,2,1,"Orginal Image","","","",True)
plot_custom(image[0,:,:,:],'','',1,2,2,"Decoded Image","","","",True)
plt.show()
```

```
1/1 [==============================] - 0s 248ms/step
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

Figure 19 - Prediction code

3.6.2. Autoencoder decoder image model (Type 2)

This model gave different train efficiency for different parameters. One run had the following results.

ID	Optimizer	Learning rates	Efficiency
1	Adagrad	0.01	60.41%
2	Adagrad	0.001	61.24%
3	Adagrad	0.0001	59.21%
4	Adagrad	0.00001	61.10%

ID	Optimizer	Learning rates	Efficiency
1	Adam	0.01	71.55%
2	Adam	0.001	60.86%
3	Adam	0.0001	69.64%
4	Adam	0.00001	61.12%

ID	Optimizer	Learning rates	Efficiency
1	Rmsprop	0.01	62%
2	Rmsprop	0.001	61.57%
3	Rmsprop	0.0001	61.16%
4	Rmsprop	0.00001	62.52%

On multiple runs, Adam, with learning rate of 0.0001 gave the best results. However, the results seem to vary for each run.

The loss value and accuracy of the model with optimizer adam and learning rate 0.0001 are as below:

Dataset	Loss	Accuracy	Val_Loss	Val_Accuracy
Train	0.0225	0.7302 - 73.65%	0.0209	0.7790 – 77.90%
Test	0.0342	0.74 – 74%	-	-

In this model, the training and validation loss obtained is given by the chart below

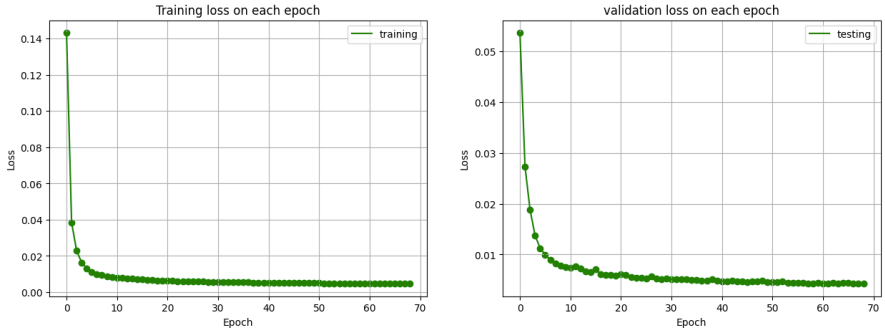


Figure 20 - Training and Validation loss

Prediction

Below is the decoded image for a given image using this model

```
[ ]  sample_image = train_data[2396]
     sample_image = np.expand_dims(sample_image,axis=0)
     image = model.predict(sample_image)
     plot_custom(sample_image[0,:,:,:],'','',1,2,1,"Orginal Image","","","",True)
     plot_custom(image[0,:,:,:],'','',1,2,2,"Decoded Image","","","",True)
     plt.show()
```

```
1/1 [==============================] - 0s 22ms/step
```

Figure 21 - Prediction code

In case of both autoencoder decoder image models that we used, we were able to see blurry image results, this is because we were trying to compress huge dimensions of image data into a smaller latent vector space.

3.6.3. CNN image model

The loss value and accuracy of the model are as below:

Dataset	Loss	Accuracy	Val_Loss	Val_Accuracy
Train	0.0135	0.5242 – 52%	0.0166	0.4050 – 40%
Test	0.0167	0.4279 – 42%	-	-

Prediction

For the randomly generated image, we displayed the actual label and predicted label. Along with the label name, the class number was also displayed to make the comparison between the actual and predicted label of the image data.

Here are some of the predictions made by the model.

3.6.4. Pretrained model

Prediction

For a given image data, the pretrained model returned the matched top 15 products (Number of products is configurable using variable N)

Since we used this pretrained model only for comparison we didn't go deep in finding singular level product match as you can see in own models. We directly used the embedding level comparison to find the similar products. Refer to sections 3.8, 3.9 and 3.10 to understand how the embeddings and

product comparison works.

```
[ ] # Finding top 15 image predictions
    top_15_image_predictions = merged_image_predictions.iloc[:15].reset_index()
    top_15_image_predictions
```

	index	Label	OriginalName	ProductName	Title	Price	image_path	label_number	values
0	9832	Women White Tops	287 image.jpg	Women White Tops_287 image.jpg	Tops for Women Basic V-Neck T-Shirt Petal Slee...	$33.99	./data/Women White Tops/287 image.jpg	22	1.000000
1	9578	Women White Tops	5 image.jpg	Women White Tops_5 image.jpg	Women's Summer Tops T Shirts Casual Ruffle Sho...	$22.99	./data/Women White Tops/5 image.jpg	22	0.884721
2	8336	Women Blue Tops	178 image.jpg	Women Blue Tops_178 image.jpg	Tops for Women Basic V-Neck T-Shirt Petal Slee...	$33.99	./data/Women Blue Tops/178 image.jpg	22	0.783664
3	9046	Women Red Tops	2 image.jpg	Women Red Tops_2 image.jpg	Womens 2022 T Shirts Petal Sleeve V Neck Casua...	$28.89	./data/Women Red Tops/2 image.jpg	22	0.759972
4	8117	Women Black Tops	223 image.jpg	Women Black Tops_223 image.jpg	Women's Summer Tops T Shirts Casual Ruffle Sho...	$22.99	./data/Women Black Tops/223 image.jpg	22	0.753783
5	9594	Women White Tops	21 image.jpg	Women White Tops_21 image.jpg	Women Casual Summer Shirts Ruched V Neck Short...	$34.99	./data/Women White Tops/21 image.jpg	22	0.743277
6	8947	Women Pink Tops	211 image.jpg	Women Pink Tops_211 image.jpg	Womens Summer Casual V Neckline Blouses Loose ...	$29.99	./data/Women Pink Tops/211 image.jpg	22	0.717851
7	9596	Women White Tops	23 image.jpg	Women White Tops_23 image.jpg	Women's Shirred V-Neck Summer Tops Short Sleev...	$33.99	./data/Women White Tops/23 image.jpg	22	0.707859
8	8976	Women Pink Tops	244 image.jpg	Women Pink Tops_244 image.jpg	Tops for Women Basic V-Neck T-Shirt Petal Slee...	$33.99	./data/Women Pink Tops/244 image.jpg	22	0.706899
9	4633	Mens Brown Shirt	54 image.jpg	Mens Brown Shirt_54 image.jpg	Womens Summer Tops Sexy V Neck Petal Sleeve Ca...	$35.99	./data/Mens Brown Shirt/54 image.jpg	22	0.706007
10	5073	Mens White Shirt	109 image.jpg	Mens White Shirt_109 image.jpg	Men's Tall Short-Sleeve Beefy T-Shirt (Pack of...	$30.77	./data/Mens White Shirt/109 image.jpg	22	0.693004
11	9047	Women Red Tops	3 image.jpg	Women Red Tops_3 image.jpg	Plus Size Tops for Women Lace Blouse Waffle Kn...	$29.99	./data/Women Red Tops/3 image.jpg	22	0.682042
12	9690	Women White Tops	126 image.jpg	Women White Tops_126 image.jpg	Womens Summer Casual Tank Tops Loose Fit Cotto...	$26.99	./data/Women White Tops/126 image.jpg	22	0.674457
13	5136	Mens White Shirt	176 image.jpg	Mens White Shirt_176 image.jpg	Mens Adaptive Magnetic Long Sleeve Button Down...	$90.99	./data/Mens White Shirt/176 image.jpg	22	0.670125
14	9783	Women White Tops	231 image.jpg	Women White Tops_231 image.jpg	Women Summer Halter Tank Tops Sleeveless Basic...	$24.99	./data/Women White Tops/231 image.jpg	22	0.667797

Figure 22 - Prediction code

3.7. Text Model

Comparison and similarity of the product title are generated using the text model. Since, in our project, text models are supplementary to the image models, the pre-trained models of sentence transformers and already available packages of tfidf vectorizers are used.

3.7.1. Sentence Transformer and its embeddings generation

Sentence Transformer can be used to generate text embeddings used for text comparison and similarity matching. This uses SBERT which is also known as Senence-BERT which uses the Triplet Siamese network to provide the necessary embeddings. This is a modification of the BERT network.

In general, sentence transformer is nothing but a python framework to produce text and image imbeddings based on given corpus of data. It has been trained in more than 100 languages with consideration on semantic search, semantic textual similarity etc

In our case, we used stsb-mpnet-base-v2 model.

The product titles are encoded to the sentence transformer to provided the text embeddings

```
# Initialize the sentence transformer and get the embeddings based on the title values
text_model = SentenceTransformer('stsb-mpnet-base-v2')
text_embeddings = text_model.encode(input.Title.values)
pickle.dump(text_model, open('/content/drive/My Drive/Intelligence beyond fashion/sentence_transformers_text_model.sav', 'wb'))
np.save('/content/drive/My Drive/Intelligence beyond fashion/sentence_transformer_text_embeddings.npy', text_embeddings)
print(text_embeddings.shape)
```

```
Downloading (…)0594e/.gitattributes: 100% 868/868 [00:00<00:00, 61.3kB/s]
Downloading (…)_Pooling/config.json: 100% 190/190 [00:00<00:00, 14.9kB/s]
Downloading (…)3f54a0594e/README.md: 100% 3.67k/3.67k [00:00<00:00, 272kB/s]
Downloading (…)54a0594e/config.json: 100% 588/588 [00:00<00:00, 46.6kB/s]
Downloading (…)ce_transformers.json: 100% 122/122 [00:00<00:00, 9.21kB/s]
Downloading pytorch_model.bin: 100% 438M/438M [00:06<00:00, 68.0MB/s]
Downloading (…)nce_bert_config.json: 100% 52.0/52.0 [00:00<00:00, 3.89kB/s]
Downloading (…)cial_tokens_map.json: 100% 239/239 [00:00<00:00, 18.7kB/s]
Downloading (…)0594e/tokenizer.json: 100% 466k/466k [00:00<00:00, 5.97MB/s]
Downloading (…)okenizer_config.json: 100% 1.19k/1.19k [00:00<00:00, 92.4kB/s]
Downloading (…)3f54a0594e/vocab.txt: 100% 232k/232k [00:00<00:00, 3.68MB/s]
Downloading (…)4a0594e/modules.json: 100% 229/229 [00:00<00:00, 17.2kB/s]
(10108, 768)
```

Figure 23 - Text embeddings code

The following code gives the topic 15 predictions for the given product title “Travel Gym Duffel Bag”

3.7.2. TFIDF Vectorizer and its embeddings generation

Term frequency inverse document frequency (TFIDF) is also used for the word embeddings but it uses the ranking factor to arrange the corpus of data by internally finding the term frequency and ranking them based on the number of times that particular word appears in the document.

Tfidf is calculated based on the multiplying term frequency with the inverse document frequency. Term frequency referes to the number of times that particular term appears in a document. Inverse document frequency referees to the inverse document frequency of the term which is log(number of documents / number of documents containing the word)

Higher the TFIDF higher the rank for that particular word.

TFIDF can be calculated as follows

Term Frequency	Number of times term appears in a document
Inverse Document Frequency	Inverse document frequency of the term which is log(number of documents / number of documents containing the word)
TFIDF	Term Frequency x Inverse Document Frequency

```
[ ] # Initalize TFIDF Vectorizer and generate a sparse matrix. Based on the sparse matrix, generate the cosine similarity matrix
    tfidfVectorizer = TfidfVectorizer()
    sparse_matrix = tfidfVectorizer.fit_transform([test_value] + input.Title)
    cosine_similarity_matrix = cosine_similarity(sparse_matrix[0,:], sparse_matrix[1:,:])
```

Figure 24 - TFIDF Vectorizer code

The product titles are encoded to the sentence transformer to provided the text embeddings. In the case of TFIDF a sparse matrix is generated which can be used to generate the cosine similarity matrix.

3.8. Embedding generation

In case of embeddings generation, the data like the text or the image is represented as vector in the lower dimensionality space.

3.8.1. Text model

The embedding generation for the text model is covered in section 3.7. This embeddings are used for the prediction of the similar product titles.

```
# Initialize the sentence transformer and get the embeddings based on the title values
text_model = SentenceTransformer('stsb-mpnet-base-v2')
text_embeddings = text_model.encode(input.Title.values)
pickle.dump(text_model, open('/content/drive/My Drive/Intelligence beyond fashion/sentence_transformers_text_model.sav', 'wb'))
np.save('/content/drive/My Drive/Intelligence beyond fashion/sentence_transformer_text_embeddings.npy', text_embeddings)
print(text_embeddings.shape)
```

```
Downloading (…)0594e/.gitattributes: 100% 868/868 [00:00<00:00, 61.3kB/s]
Downloading (…)_Pooling/config.json: 100% 190/190 [00:00<00:00, 14.9kB/s]
Downloading (…)3f54a0594e/README.md: 100% 3.67k/3.67k [00:00<00:00, 272kB/s]
Downloading (…)54a0594e/config.json: 100% 588/588 [00:00<00:00, 46.6kB/s]
Downloading (…)ce_transformers.json: 100% 122/122 [00:00<00:00, 9.21kB/s]
Downloading pytorch_model.bin: 100% 438M/438M [00:06<00:00, 68.0MB/s]
Downloading (…)nce_bert_config.json: 100% 52.0/52.0 [00:00<00:00, 3.89kB/s]
Downloading (…)cial_tokens_map.json: 100% 239/239 [00:00<00:00, 18.7kB/s]
Downloading (…)0594e/tokenizer.json: 100% 466k/466k [00:00<00:00, 5.97MB/s]
Downloading (…)okenizer_config.json: 100% 1.19k/1.19k [00:00<00:00, 92.4kB/s]
Downloading (…)3f54a0594e/vocab.txt: 100% 232k/232k [00:00<00:00, 3.68MB/s]
Downloading (…)4a0594e/modules.json: 100% 229/229 [00:00<00:00, 17.2kB/s]
(10108, 768)
```

Figure 25 - Sentence transformer code

3.8.2. Image model

In case image model, similar to the text model, once the model is created, numpy is used to convert the model into numerical array data which is used further for processing.

```
[ ] # Here, np.expand is used to change the 3 dimension data into 4 dimension
    cnn_image_model.predict(np.expand_dims(X_test[0],axis = 0)).round(2)

    1/1 [==============================] - 0s 190ms/step
    array([[0.  , 0.  , 0.01, 0.  , 0.  , 0.  , 0.  , 0.  , 0.2 , 0.  , 0.  ,
            0.  , 0.  , 0.  , 0.03, 0.14, 0.  , 0.  , 0.  , 0.03, 0.05, 0.15, 0.1 ,
            0.  , 0.  , 0.  , 0.17, 0.04, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ,
            0.  , 0.  , 0.  , 0.01, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.04]],
          dtype=float32)

[ ] np.argmax(cnn_image_model.predict(np.expand_dims(X_test[0],axis = 0)).round(2))

    1/1 [==============================] - 0s 22ms/step
    8
```

Figure 26 - Embedding generation

3.9. Similarity Matching

Similarity matching is done based on the embeddings generated. This is done differently for both the text and image, later they are combined. However, cosine similarity is mostly used to determine the similarity. Cosine values provide the angular difference between two different embeddings in the vector space. It can be seen that Euclidean distance can also be used however it in the contextual data it is found that, cosine values provide more accurate results.

3.9.1. Text model

3.9.1.1. Sentence Transformer

In this case, once the text embeddings are generated, the embeddings are then normalized.

```
# Normalize Embeddings
def normalize_embeddings(embeddings):
    for embedding in embeddings:
        temp = np.linalg.norm(embedding)
        embedding/=temp
    return embeddings
```

Figure 27 - Normalize embeddings

The given input text is encoded or converted to the embedding format and cosine values are generated based on the model embeddings available. Based on the predictions are found.

```
[ ] # Function to predict based on the cosine value
    def cosine_value_predictions(embeddings, curr_embedding):
        cosine_value = np.matmul(embeddings, curr_embedding.T).T
        reshape_data = np.reshape(cosine_value > 0.0,(len(embeddings),))
        input['values'] = np.reshape(cosine_value,(len(embeddings),))
        return input[reshape_data].sort_values(by='values', ascending=False)
```

Figure 28 - Cosine value predictions

```
[ ] # Get the text predictions based on the cosine similarity
    text_predictions = cosine_value_predictions(sentence_transformer_text_embeddings, input_text_embedding)
```

```
# Get the top 15 predictions
text_predictions_top_15 = text_predictions.iloc[:15].reset_index()
text_predictions_top_15
```

	index	Label	OriginalName	ProductName	Title	Price	image_path	label_number	values
0	7045	Shoulder bags	297 image.jpg	Shoulder bags_297 image.jpg	Mens Duffle/Gym Bag	$36.82	./data/Shoulder bags/297 image.jpg	22	2.720549
1	2653	Duffel bags	98 image.jpg	Duffel bags_98 image.jpg	Gym Bag for Men and Women Waterproof Gym Duffl...	$25.99	./data/Duffel bags/98 image.jpg	22	2.677719
2	2852	Duffel bags	298 image.jpg	Duffel bags_298 image.jpg	Bosidu Gym Duffle Bag for Men & Women, Large S...	$39.99	./data/Duffel bags/298 image.jpg	22	2.623933
3	2749	Duffel bags	194 image.jpg	Duffel bags_194 image.jpg	Sport Gym Bag Large Travel Duffel Bag with Sho...	$38.99	./data/Duffel bags/194 image.jpg	22	2.623302
4	2840	Duffel bags	286 image.jpg	Duffel bags_286 image.jpg	Gym Duffle Bag, Goodsnetic Sports Gym Bag Spor...	$28.99	./data/Duffel bags/286 image.jpg	22	2.598163
5	2820	Duffel bags	266 image.jpg	Duffel bags_266 image.jpg	Travel Gym Bag for Men Women Waterproof Sports...	$34.99	./data/Duffel bags/266 image.jpg	22	2.596679
6	2569	Duffel bags	14 image.jpg	Duffel bags_14 image.jpg	Gym Duffle Bag Waterproof Sports Duffel Bags T...	$54.86	./data/Duffel bags/14 image.jpg	22	2.596404
7	2622	Duffel bags	67 image.jpg	Duffel bags_67 image.jpg	Sports Gym Bag with Shoes Compartment Travel D...	$36.99	./data/Duffel bags/67 image.jpg	22	2.578816
8	2598	Duffel bags	43 image.jpg	Duffel bags_43 image.jpg	Gym Duffle Bag for Men Sports Travel Backpack ...	$36.99	./data/Duffel bags/43 image.jpg	22	2.574959
9	2813	Duffel bags	259 image.jpg	Duffel bags_259 image.jpg	Sports Gym Bag, Travel Duffel Bag with Wet Poc...	$26.99	./data/Duffel bags/259 image.jpg	22	2.551054
10	2697	Duffel bags	142 image.jpg	Duffel bags_142 image.jpg	OMBRY Gym Duffle Bag Womens Weekender Travel B...	$35.99	./data/Duffel bags/142 image.jpg	22	2.547934
11	2578	Duffel bags	23 image.jpg	Duffel bags_23 image.jpg	Gym Duffle Bag Waterproof Leather Large Sports...	$59.99	./data/Duffel bags/23 image.jpg	22	2.545250
12	2649	Duffel bags	94 image.jpg	Duffel bags_94 image.jpg	Gym Bag Sports Duffel Bag with Wet Pocket & Sh...	$36.99	./data/Duffel bags/94 image.jpg	22	2.531597
13	2738	Duffel bags	183 image.jpg	Duffel bags_183 image.jpg	Suruid Gym Bag, Travel Duffel Bag, Sports Tote...	$31.99	./data/Duffel bags/183 image.jpg	22	2.520691
14	7739	Tote bags	266 image.jpg	Tote bags_266 image.jpg	Travel Gym Bag for Women, LANBX Tote Bag Carry...	$32.99	./data/Tote bags/266 image.jpg	22	2.509548

Figure 29 - Text predictions

3.9.1.2. TFIDF Vectorizer

TFIDF Vectorizer is also similar to sentence transformer. Once the tfidf vectorizer is trained with product titles, a sparse matrix with the necessary embeddings are generated.
This embeddings are fed into similar kind of cosine determining function to find the relevant cosine values. As a result a cosine similarity matrix is generated.

```
# Initalize TFIDF Vectorizer and generate a sparse matrix. Based on the sparse matrix, generate the cosine similarity matrix
tfidfVectorizer = TfidfVectorizer()
sparse_matrix = tfidfVectorizer.fit_transform([test_value] + input.Title)
cosine_similarity_matrix = cosine_similarity(sparse_matrix[0,:], sparse_matrix[1:,:])
```

```
[ ] # Get the text predictions based on the cosine similarity
    text_predictions = pd.DataFrame({'Title': input.Title[1:], 'values': cosine_similarity_matrix[0]})
    text_predictions = text_predictions.sort_values('values', ascending=False)
```

Figure 30 - TFIDF Vectorizer

```
[ ] # Function to predict based on the cosine value
    def cosine_value_predictions(embeddings, curr_embedding):
        cosine_value = np.matmul(embeddings, curr_embedding.T).T
        reshape_data = np.reshape(cosine_value > 0.0,(len(embeddings),))
        input['values'] = np.reshape(cosine_value,(len(embeddings),))
        return input[reshape_data].sort_values(by='values', ascending=False)
```

Figure 31 - Cosine value predictions

This cosine similarity matrix is further used to find the similar product titles for a given product title.
As we can see in the above code, the test value is also needed to be added to generate the sparse matrix to find the proper cosine similarity values.

```
# Get the top 15 text predictions
text_predictions_top_15 = text_predictions.iloc[:15].reset_index()
text_predictions_top_15
```

	index	Title	values
0	2565	Amazon Basics Large Travel Luggage Duffel Bag ...	0.686731
1	2683	Amazon Basics Packable Travel Gym Duffel Bag -...	0.605938
2	2745	Amazon Basics Packable Travel Gym Duffel Bag -...	0.582323
3	2556	Amazon Basics Large Travel Luggage Duffel Bag ...	0.582273
4	2583	Amazon Basics Large Travel Luggage Duffel Bag ...	0.539272
5	31	Amazon Basics Ultralight Portable Packable Day...	0.469783
6	2790	Amazon Basics Ripstop Rolling Travel Luggage D...	0.400524
7	5331	Amazon Basics 17.3-Inch Laptop Case Bag, Fits ...	0.370242
8	712	Amazon Collection Sterling Silver bracelet	0.343969
9	85	Laptop Backpack Women 15.6 Inch Travel Backpac...	0.342678
10	3558	Mens Basics Emblem Graphic Hoodie	0.322821
11	67	Laptop Backpack 15.6 Inch Travel Backpack with...	0.318119
12	174	Travel Backpack for Men Women Laptop Backpack ...	0.316891
13	107	Laptop Backpack 15.6 Inch Travel Backpack for ...	0.315726
14	41	Vaupan Travel Backpack for men women, Carry On...	0.312146

Figure 32 - Top 15 predicted values

```
related_top_15_text_data = text_predictions_top_15.Title.values
related_top_15_text_data = [text.strip() for text in related_top_15_text_data]
related_top_15_text_data
```

```
['Amazon Basics Large Travel Luggage Duffel Bag - Black',
 'Amazon Basics Packable Travel Gym Duffel Bag - 23 Inch, Black',
 'Amazon Basics Packable Travel Gym Duffel Bag - 27 Inch, Black',
 'Amazon Basics Large Travel Luggage Duffel Bag - Red',
 'Amazon Basics Large Travel Luggage Duffel Bag - Navy Blue',
 'Amazon Basics Ultralight Portable Packable Day Pack',
 'Amazon Basics Ripstop Rolling Travel Luggage Duffle Bag With Wheels - 35 Inch, Blue',
 'Amazon Basics 17.3-Inch Laptop Case Bag, Fits Dell, HP, ASUS, Lenovo, MacBook Pro and more, Black',
 'Amazon Collection Sterling Silver bracelet',
 'Laptop Backpack Women 15.6 Inch Travel Backpack for Women Waterproof College School Backpack for Work Business, Black',
 'Mens Basics Emblem Graphic Hoodie',
 'Laptop Backpack 15.6 Inch Travel Backpack with USB Charging Port Large Waterproof Computer Backpack College Backpack for Work Business School Carry on Backpack for Men and Women, Black',
 'Travel Backpack for Men Women Laptop Backpack 15.6 Inch Waterproof Backpack for School Carry On Luggage Backpack Flight Approved Work College Computer Backpack,Black',
 'Laptop Backpack 15.6 Inch Travel Backpack for Women Men Waterproof Computer Backpack with USB Charging Port Large Carry On Backpack for Work School, Black',
 'Vaupan Travel Backpack for men women, Carry On Backpack, Durable Water Resistant College School Backpack laptop bag, Large Casual Daypack Hiking Backpack (Black)']
```

Figure 33 - Top 15 text predictions

3.9.2. Image model

In case of image model, once the embeddings are obtained, inorder to find the similar images or products, it is highly essential to cluster or group similar products together so when a new input image / embedding is given, we can find the cluster or group this new input image / embedding belongs to.

Two main methods of clustering or grouping is by using:

1. KMeans and KNN Similarity Matching
2. Cosine Similarity Matching

3.9.2.1. KMeans and KNN Similarity Matching

This is a four step process as follows:

1. At first the necessary features are extracted from the model layers

```
model = tf.keras.models.load_model("/content/drive/My Drive/Intelligence beyond fashion/autoencoder_decoder_first_image_model.h5")

"""
Extract features out of the model
@param - trained_model - trained model
@param - image_data - image data
@param - layer - the layer from which the features should be obtained
@return encoded array
"""
def extract_features(trained_model, image_data, layer=4):
    encoded = K.function([trained_model.layers[0].input], [trained_model.layers[layer].output])
    encoded_array = encoded([image_data])[0]
    pooled_array = encoded_array.max(axis=-1)
    return encoded_array

# Input the layer below according to the model
# Here only for autoencoder_decoder_first_image_model and hence 9th layer should be correct
encoded = extract_features(model, train_data[:10], 9)
```

Figure 34 - Feature extraction

```
# Display the encoded detais for three different images
for index in [5,7,8]:
    plt.figure(figsize=(15,3))
    plot_custom(train_data[index],'','',1,4,1,"Original Image","","",'',True)
    plot_custom(encoded[index].mean(axis=-1),'','',1,4,2,"Encoded Mean","","",'',True)
    plot_custom(encoded[index].max(axis=-1),'','',1,4,3,"Encoded Std","","",'',True)
    plot_custom(encoded[index].std(axis=-1),'','',1,4,4,"Encoded Std","","",'',True)
    plt.show()
```

Figure 35 - Encoded mean and standard deviation

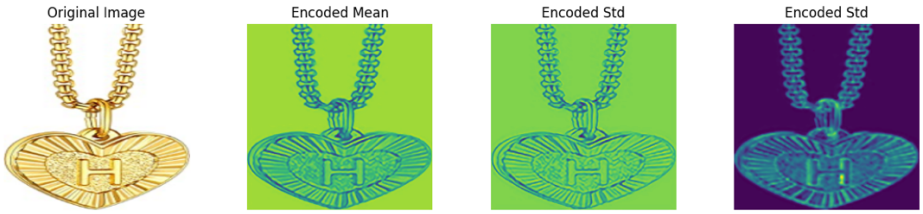


Figure 36 - Encoded mean and standard deviation

As a result, encoded and reshaped data is obtained

2. Use K-Means to cluster the encoded and reshaped data obtained

   Number of clusters is determined in an iterative manner.

```
num_clusters = [3, 4, 5, 6, 7]
for k in num_clusters:
    print("if Number of clusters: "+str(k))
    kmeans = KMeans(n_clusters=k, random_state=0).fit(X_encoded_reshaped_data)
    labels = kmeans.labels_
    centroids = kmeans.cluster_centers_
    plt.figure(figsize=(10,5))
    plt.subplot(1,1,1)
    plt.scatter(values[:,0], values[:,1], c=kmeans.labels_.astype(float), s=50, alpha=0.5)
    plt.scatter(centroids[:, 0], centroids[:, 1], c=None, s=50)
    plt.show()
    for row in range(k):
        iter=0
        plt.figure(figsize=(13,3))
        for i, iterator in enumerate(labels):
            print(i)
            if iterator == row:
                img = cv2.imread(lisp[i])
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                plot_(img,"","",1,6,iter+1,"cluster="+str(row),"","","",True)
                iter+=1
            if iter>=5: break
        plt.show()
```

Figure 37 - Number of clusters determination

Based on the selected number of clusters, the KMeans is trained and the clusters are formed

```
# Training the kmeans. Also, finding the labels and centroids of the clusters
kmeans = KMeans(n_clusters = len(main_labels), random_state=0).fit(X_encoded_reshaped_data)
cluster_labels = kmeans.labels_
cluster_centers = kmeans.cluster_centers_
joblib.dump(kmeans, '/content/drive/My Drive/Intelligence beyond fashion/kmeans_model.pkl')
```

Figure 38 - Cluster data formation

3. K-NN is used to find the nearest neighbours

```
knn = KNeighborsClassifier(n_neighbors=9, algorithm='ball_tree',n_jobs=-1)
knn.fit(np.array(features), np.array(cluster_labels))
joblib.dump(knn, '/content/drive/My Drive/Intelligence beyond fashion/knn_model.pkl')
```

Figure 39 - KNN

4. Predictions are found based on the KNN data and the features obtained

```
feature = K.function([model.layers[0].input],[model.layers[9].output])
output = feature([img_data])
output = np.array(output).flatten().reshape(1,-1)
res = knn.kneighbors(output.reshape(1,-1),return_distance=True,n_neighbors=N)
display_results(img,list(res[1][0])[1:])
```

Figure 40 - Predictions

The KMeans and KNN similarity matching method didn't seem promising compared to the Cosine similarity matching method.

3.9.2.2. Cosine Similarity Matching

In this method, the cosine values between the different embeddings are generated and stored.

```
# Function to predict based on the cosine value
def cosine_value_predictions(embeddings, curr_embedding):
    cosine_value = np.matmul(embeddings, curr_embedding.T).T
    reshape_data = np.reshape(cosine_value > 0.0,(len(embeddings),))
    input['values'] = np.reshape(cosine_value,(len(embeddings),))
    return input[reshape_data].sort_values(by='values', ascending=False)
```

Figure 41 - Cosine similarity predictions

With the input image/embedding, again, the cosine values among the different embeddings are generated to find the angles in the vector space. The angles thus obtained are arranged in ascending order. The lower the angle, it is how closer the embeddings belong. That means how similar the images are.

```
merged_image_predictions = cosine_value_predictions(own_model_embeddings, image_embedding)
```

```
top_15_image_predictions = merged_image_predictions.iloc[:15].reset_index()
top_15_image_predictions
```

	index	Label	OriginalName	ProductName	Title	Price	image_path	label_number	values
0	1480	Cargo Shorts	247 image.jpg	Cargo Shorts_247 image.jpg	Women's Hiking Cargo Shorts Pockets Waterproof...	$34.99	./data/Cargo Shorts/247 image.jpg	27	1.0
1	6602	Running Shoes	51 image.jpg	Running Shoes_51 image.jpg	Women's Fresh Foam Arishi V3 Running Shoe	$99.99	./data/Running Shoes/51 image.jpg	27	1.0
2	8985	Women Pink Tops	254 image.jpg	Women Pink Tops_254 image.jpg	Plus Size Workout Tank Tops for Women - Runnin...	$39.58	./data/Women Pink Tops/254 image.jpg	27	1.0
3	2600	Duffel bags	45 image.jpg	Duffel bags_45 image.jpg	Lekesky Travel Duffle Bag Women Weekend Overni...	$39.99	./data/Duffel bags/45 image.jpg	27	1.0
4	6144	Pullover hoodies	213 image.jpg	Pullover hoodies_213 image.jpg	Womens Full Zipper Hoodies Fleece Lined Collar...	$79.35	./data/Pullover hoodies/213 image.jpg	27	1.0
5	5542	Messenger bags	314 image.jpg	Messenger bags_314 image.jpg	Extra Slim Crossbody Purse Men and Women Small...	$35.99	./data/Messenger bags/314 image.jpg	27	1.0
6	4800	Mens Grey Shirt	72 image.jpg	Mens Grey Shirt_72 image.jpg	Men's Golf Shirts Short Sleeve Dry Fit Moistur...	$31.99	./data/Mens Grey Shirt/72 image.jpg	27	1.0
7	169	Backpacks	175 image.jpg	Backpacks_175 image.jpg	YAMTION Backpack Men and Women, School Backpac...	$44.99	./data/Backpacks/175 image.jpg	27	1.0
8	171	Backpacks	178 image.jpg	Backpacks_178 image.jpg	Travel Backpack for Women, Flight Approved Car...	$44.99	./data/Backpacks/178 image.jpg	27	1.0
9	6532	Rings	322 image.jpg	Rings_322 image.jpg	18K Gold Plated 7 Cubic Fungus Anti-Bacteria S...	$22.50	./data/Rings/322 image.jpg	27	1.0
10	371	Baseball hats	77 image.jpg	Baseball hats_77 image.jpg	2 Pack Full Cotton Casual Hats Baseball Caps f...	$28.95	./data/Baseball hats/77 image.jpg	27	1.0
11	7783	Tote bags	310 image.jpg	Tote bags_310 image.jpg	Cinar Tote Bag, Canvas Tote Bag For Women With...	$18.18	./data/Tote bags/310 image.jpg	27	1.0
12	566	Beanies	201 image.jpg	Beanies_201 image.jpg	Unisex-Adult Mountain High Fleece-Lined Beanie	$44.99	./data/Beanies/201 image.jpg	27	1.0
13	4811	Mens Grey Shirt	87 image.jpg	Mens Grey Shirt_87 image.jpg	Men's Stretch Muscle Tshirts Long Sleeve Knit ...	$28.89	./data/Mens Grey Shirt/87 image.jpg	27	1.0
14	7780	Tote bags	307 image.jpg	Tote bags_307 image.jpg	3pcs Extra Large 26.5 gal Storage Bags with St...	$30.99	./data/Tote bags/307 image.jpg	27	1.0

Figure 42 - Top 15 image predictions

3.10. Combined Predictions

Since the text embeddings and image embeddings thus obtained are nd arrays, for combined predictions, it was directly combined by multiplying the embedding data

```
# Combine the predictions and get the top specified predictions
N = 15
source_data['values'] = np.reshape(np.matmul(source_text_and_images, np.concatenate([image_data, text_data]).T).T, (len(source_text_and_images),))
top_n_predictions = source_data[np.reshape((np.matmul(source_text_and_images, np.concatenate([image_data, text_data]).T).T > 0.0), (len(source_text_and_images),))].sort_values(by='values', ascending=False).iloc[:N
top_n_predictions
```

Figure 43 - Combined predictions code

Based on this, again cosine similarity method is used to calculate similar products.

```
# Combine the predictions and get the top specified predictions
N = 15
source_data['values'] = np.reshape(np.matmul(source_text_and_images, np.concatenate([image_data, text_data]).T).T, (len(source_text_and_images),))
top_n_predictions = source_data[np.reshape((np.matmul(source_text_and_images, np.concatenate([image_data, text_data]).T).T > 0.0), (len(source_text_and_images),))].sort_values(by='values', ascending=False).iloc[
top_n_predictions
```

	index	Label	OriginalName	ProductName	Title	Price	image_path	label_number	values
0	8974	Women Pink Tops	242 image.jpg	Women Pink Tops_242 image.jpg	Lite Women's Lightweight & Athletic Philosophy...	$50.19	./data/Women Pink Tops/242 image.jpg	27	3.035720
1	6989	Shoulder bags	240 image.jpg	Shoulder bags_240 image.jpg	Small Crossbody Bags for Women Purses Fashion ...	$29.99	./data/Shoulder bags/240 image.jpg	27	2.955401
2	6836	Shoulder bags	85 image.jpg	Shoulder bags_85 image.jpg	Crossbody Bag for Women PU Leather Shoulder Ba...	$23.19	./data/Shoulder bags/85 image.jpg	27	2.932648
3	5888	Puffer Jackets	21 image.jpg	Puffer Jackets_21 image.jpg	Women's Hooded Packable Ultra Light Weight Sho...	$95.89	./data/Puffer Jackets/21 image.jpg	27	2.929463
4	7139	Sneakers	16 image.jpg	Sneakers_16 image.jpg	Women's Energycloud Lightweight Slip On Walkin...	$44.99	./data/Sneakers/16 image.jpg	27	2.891430
5	6557	Running Shoes	6 image.jpg	Running Shoes_6 image.jpg	Women's Energycloud Lightweight Slip On Walkin...	$44.59	./data/Running Shoes/6 image.jpg	27	2.891430
6	7076	Shoulder bags	328 image.jpg	Shoulder bags_328 image.jpg	Mini Purse Small Shoulder Bags for Women Trend...	$59.88	./data/Shoulder bags/328 image.jpg	27	2.885703
7	6924	Shoulder bags	174 image.jpg	Shoulder bags_174 image.jpg	Crossbody Bag For Women Waterproof Lightweight...	$33.80	./data/Shoulder bags/174 image.jpg	27	2.846617
8	6757	Shoulder bags	2 image.jpg	Shoulder bags_2 image.jpg	Crossbody Bags for Women Small Shoulder Bag Ha...	$41.99	./data/Shoulder bags/2 image.jpg	27	2.838963
9	7736	Tote bags	263 image.jpg	Tote bags_263 image.jpg	Women's Avalon Small Tote Bag	$54.90	./data/Tote bags/263 image.jpg	27	2.834651
10	5508	Messenger bags	280 image.jpg	Messenger bags_280 image.jpg	Women's Sebastian Crossbody, Super Light, Dura...	$106.99	./data/Messenger bags/280 image.jpg	27	2.833181
11	3420	Graphic hoodies	142 image.jpg	Graphic hoodies_142 image.jpg	Womens Relaxed Fit Midweight Logo Sleeve Graph...	$84.99	./data/Graphic hoodies/142 image.jpg	27	2.827038
12	1409	Cargo Shorts	176 image.jpg	Cargo Shorts_176 image.jpg	Womens 11-inch Relaxed Cargo Short	$45.70	./data/Cargo Shorts/176 image.jpg	27	2.801595
13	6788	Shoulder bags	33 image.jpg	Shoulder bags_33 image.jpg	Shoulder Bag for Women Small Handbag with Zipp...	$26.99	./data/Shoulder bags/33 image.jpg	27	2.791205
14	7022	Shoulder bags	274 image.jpg	Shoulder bags_274 image.jpg	Lightweight Gym Tote Bag Waterproof Sports Han...	$38.99	./data/Shoulder bags/274 image.jpg	27	2.785494

Figure 44 - Top 15 combined predictions

3.11. Trend Analysis

3.11.1. Introduction

The objective of this project is to explore Twitter for tweets related to sneaker culture. The tweets will be filtered using hashtags such as #sneakerhead, #shoestyle, #iloveshoes, #NikeBlazer, yeezy, #puma, #airmax, #nikeairmax, jordans, #nike, #adidas, @Nike, @adidas, @PUMA, @newbalance, @JDSports, adidas, nike, #airjordan, #jordans, #jdsports, and filtered out retweets. The tweets will be analyzed to find the topic, most frequent words, hashtags, and sources of tweets.

3.11.2. Dependencies

The following packages need to be installed before proceeding with the code:

• tweepy==3.10.0, pandas, re, html, collections, matplotlib, wordcloud, nltk stopwords

3.11.3. Methodology

- Import the necessary packages.
- Construct the Twitter API with the appropriate authentication keys.
- Define the search query to filter the tweets based on the hashtags and other filters.
- Collect the tweets using the search query and store them in a list.
- Create a pandas dataframe to store the information extracted from the tweets.
- Save the dataframe to a CSV file for future use.
- Clean the text data using a regular expression to remove any unwanted characters, URLs, tags, etc.
- Tokenize the cleaned text data and remove stopwords.
- Perform a word frequency analysis to find the most common words used in the tweets.
- Visualize the results using a word cloud.
- Implementing Topic Modelling techniques i.e., LDA and NMF
- Implementation of two topic modelling techniques:
    - Latent Dirichlet Allocation
    - Non Negative Matrix Factorization

Topic modelling - It is a technique used to extract latent topics from a corpus of documents, where each document may contain multiple topics. The goal of topic modeling is to identify the underlying themes and patterns in the text data.

LDA - The Latent Dirichlet Allocation is called LDA. Natural Language Processing (NLP) uses this statistical model to find topics in a corpus (a large collection of text documents) and classify them appropriately. Each

document in the corpus is assumed to be a combination of different topics, and each topic is assumed to be a fixed vocabulary distribution of words. The program tries to determine the most likely topic for each document and the distribution of terms within each topic. In NLP, LDA is a popular topic modeling technique that has been applied to various tasks such as document classification, information retrieval, and recommendation engines.

NMF - On the other hand, NMF is a matrix factorization technique that assumes that the documents in a corpus are represented as a linear combination of a small number of latent topics, where each topic is a non-negative linear combination of words. It aims to factorize the document-term matrix into two low-rank non-negative matrices representing the topic and word distributions. NMF can be used to identify the topics in a given corpus of documents by iteratively updating the topic and word distributions until convergence.
Both LDA and NMF are unsupervised algorithms, meaning that they do not require any prior knowledge about the topics in the corpus. They are widely used in various applications such as text classification, recommendation systems, and information retrieval.
In general, LDA is more interpretable and produces more coherent topics than NMF. However, NMF is faster and more robust to noisy data. The choice between LDA and NMF depends on the specific requirements of the application and the characteristics of the data.

3.11.4. Results

A total of 2000 tweets were obtained using the search query, and a dataframe was created to store the information extracted from these tweets. The dataframe contains information about the user name, user description, tweet text, hashtags, and source of the tweet.
After cleaning the text data and removing stopwords, a word frequency analysis was performed. The most common words used in the tweets are "Nike," "Adidas", "Air", "Chili", "Air Jordan."

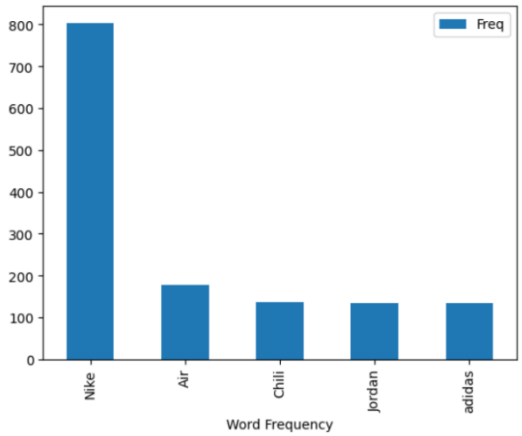


Figure 45 - Word frequency

A word cloud was also created to visualize the most common words used in the tweets. The word cloud shows that the most common words used in the tweets are "sneaker," "Nike," "Adidas," "shoes," "Air Jordan," "Yeezy," "Puma," and "Jordans."

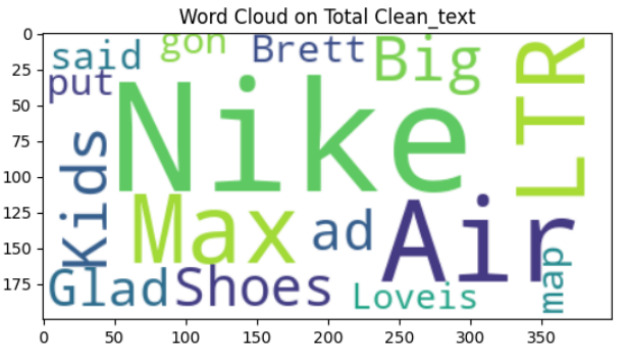
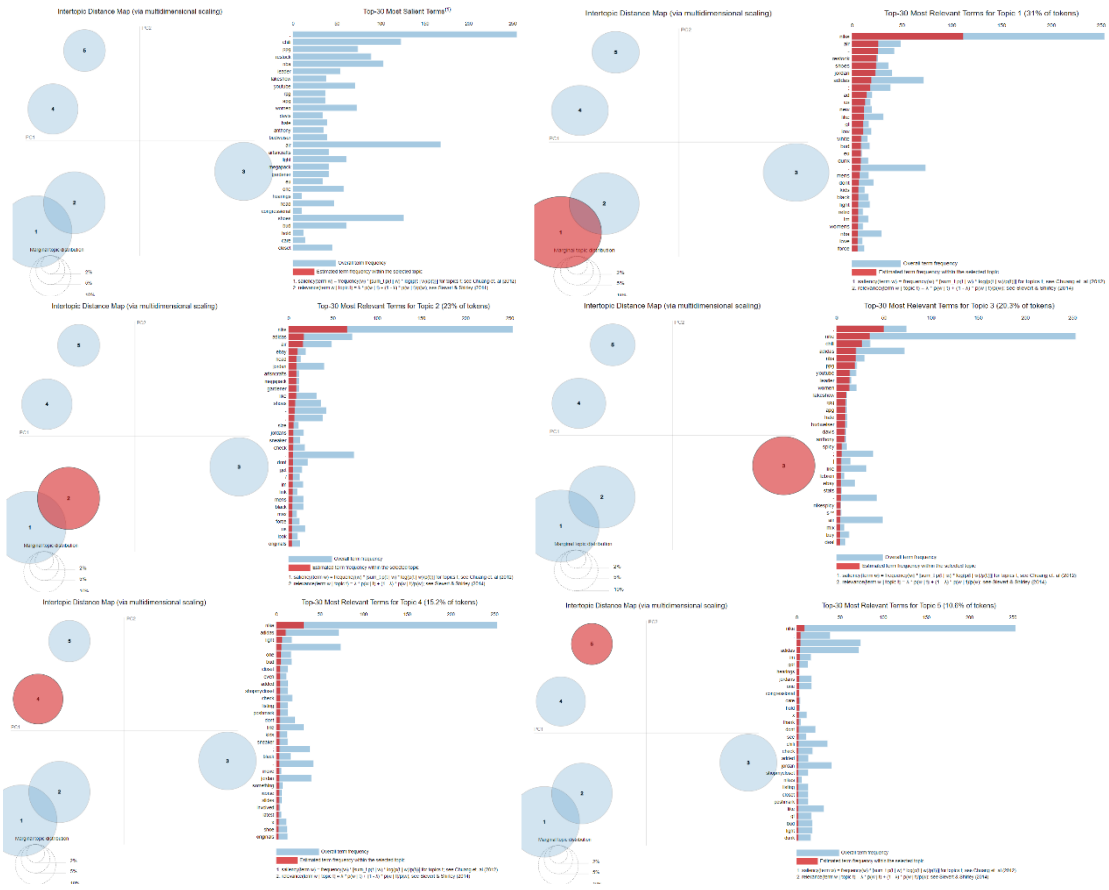
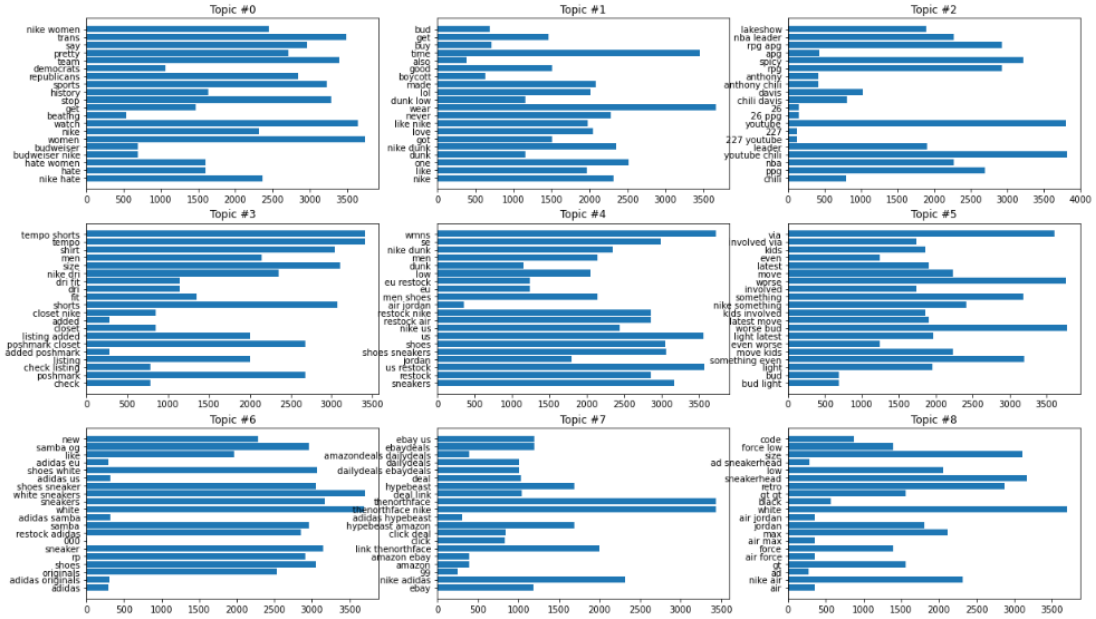


Figure 46 - Word cloud

LDA – The results obtained from LDA using PyLDAVis can be very helpful in analyzing and interpreting the topics generated by the algorithm. analysts can gain insights into the topics generated by LDA, such as understanding which terms are most relevant to each topic, which topics are more dominant in the corpus, and which topics are most similar to each other.
Overall, PyLDAVis is a powerful tool that can help analysts understand and interpret the results of LDA topic modelling.



NMF – In the context of tweets, NMF can be used to identify and extract underlying topics from a large set of tweets.



3.12. Web Application in ReactJS

This provides an overview of a web application built using React JS. The application consists of several components that make up different sections of the website. Here, we will go through each component and describe their functionality.

3.12.1. Components

Header

The Header component is responsible for rendering the website's header, including the navigation menu. The header contains links to navigate through various pages within the application, such as Home, Contact, About Us, and others.

Footer

The Footer component renders the footer section of the website. This section typically contains copyright information, links to social media profiles, and other relevant information.

Home

The Home component is the landing page of the application. This page showcases the main content and features of the website.

Contact

The Contact component displays a contact form that allows users to get in touch with the website owner. It includes input fields for the user's first name, last name, email address, phone number, and a message. Upon submission, the form data is sent to the specified endpoint for further processing.

About

The About component is responsible for displaying information about the website or organization. It typically contains a description, mission statement, and team member profiles.

Trends

The Trends component is a page that displays current trends, such as ongoing matches, events, or other relevant data. It accepts a setFavouritesData function as a prop, which is used to update the favorites list.

MatchTrends

The MatchTrends component is a page that displays trend information for a specific match or event. Users can view detailed information about the match, including team statistics, player performance, and more.

Favourites

The Favourites component is responsible for displaying a list of user favorites. It accepts a favouritesData prop, which is an array containing the user's favorite items.

3.12.2. App Component and Routing

The App component serves as the main container for the application. It sets up the routing for the website using the react-router-dom package, which allows users to navigate between pages. The App component also maintains state for the user's favorites, which is passed down as props to the Trends and Favourites components.

3.12.3. Routes

The following routes are defined within the App component:

1. Home (ROUTE_PATHS.HOME): Renders the Home component.
2. Contact (ROUTE_PATHS.CONTACT): Renders the Contact component.
3. About Us (ROUTE_PATHS.ABOUT_US): Renders the About component.
4. Current Match (ROUTE_PATHS.CURRENT_MATCH): Renders the Trends component.
5. Trend Match (ROUTE_PATHS.TREND_MATCH): Renders the MatchTrends component.
6. Favourites (ROUTE_PATHS.FAVOURITES): Renders the Favourites component.

3.12.4. Constants

The ROUTE_PATHS object holds the route path constants that are used throughout the application:

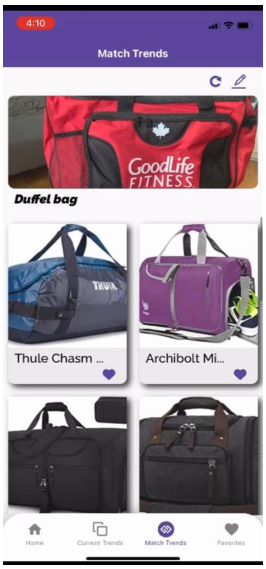
```
export const ROUTE_PATHS = {
  HOME: "/",
  CONTACT: "/contact",
  ABOUT_US: "/about-us",
  CURRENT_MATCH: "/current-match",
  TREND_MATCH: "/trend-match",
  FAVOURITES: "/favourites",
};
```

3.12.5. Wrapping Up

This React-based web application is made up of several components that interact with each other to create a cohesive user experience. Each component is responsible for rendering a specific part of the website, and the App component

3.13. Mobile Application in React-Native

The React Native Trends App is a mobile application designed to showcase and manage trends. The app consists of four main screens: Home, Current Trends, Match Trends, and Favorites. Each screen displays specific information related to trends, allowing users to explore, compare, and save their favorite trends. This documentation provides an overview of the app's structure, the components used, and their functionalities.



3.13.1. App Structure

The app is built using React Native and consists of three main files:

1. App.js - The entry point of the application, which sets up the navigation structure and styles.
2. CurrentTrends.js - A component displaying a list of current trends.
3. Favorites.js - A component allowing users to browse and manage their favorite trends.

3.13.2. Components

Home

The Home component serves as the landing screen of the app. While not provided in the code snippets above, it is likely to contain an overview or summary of the latest trends, along with navigation options to access other screens.

Current Trends

The CurrentTrends component displays a list of current trends in a grid format using FlatList. Each item in the list is rendered using a Card component, which includes an image, title, and category. The data for the current trends is imported from a local data file (../../Data/data).

Figure 47 - Current trends

Match Trends

The MatchTrends component is not provided in the code snippets above, but it is likely to be responsible for comparing trends, allowing users to view how different trends relate or match with each other.

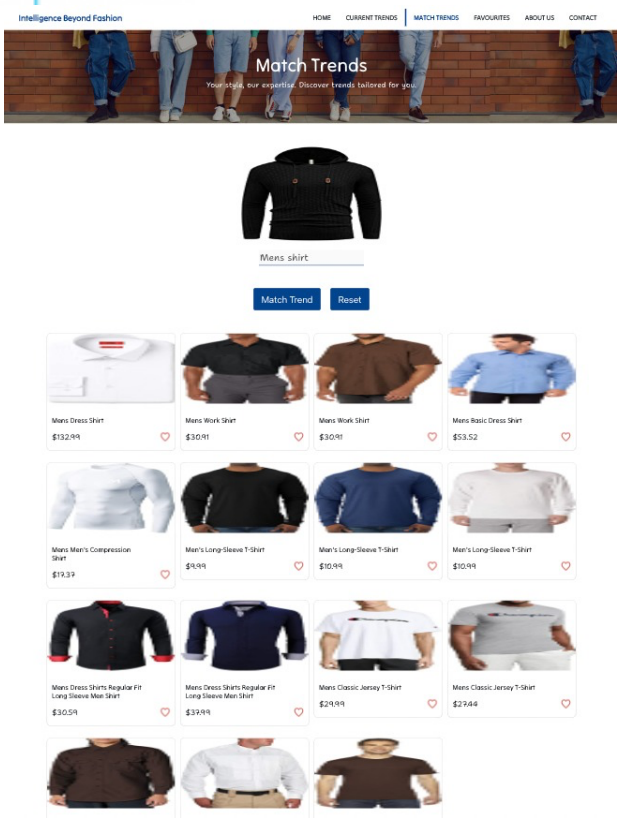


Figure 48 - Match trends

Favorites

The Favorites component allows users to browse and manage their favorite trends. It displays a list of favorite trends using a FlatList with a Card component similar to the CurrentTrends component. Users can interact with the trends and perform actions such as adding or removing favorites.

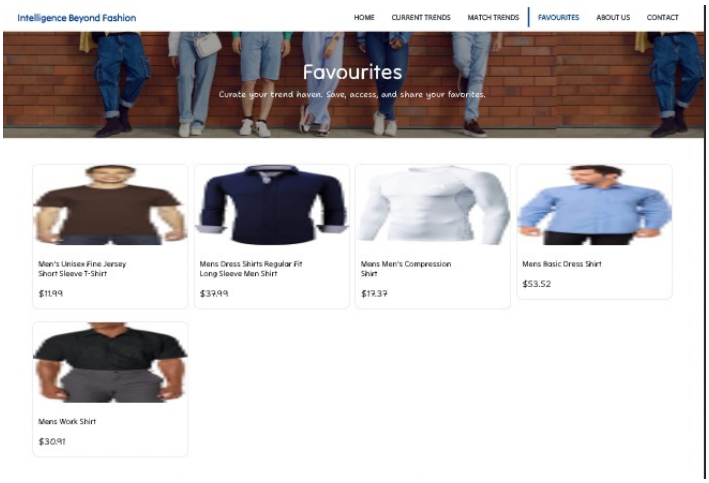


Figure 49 - Favorites

Navigation

The app uses the react-navigation library to handle navigation between the different screens. A bottom tab navigator is set up in App.js with four screens: Home, Current Trends, Match Trends, and Favorites. Each screen is configured with a custom icon, header styles, and a corresponding component.

Fonts and Styles

Custom fonts are loaded using the expo-font package, and the app uses the useFonts hook to load these fonts asynchronously. An AppLoading component is shown while the fonts are being loaded.
The app's header and bottom tab bar styles are defined in headerStyles and tabBarStyle objects in App.js. These styles are applied to each screen in the bottom tab navigator.

Usage

To use the app, users can navigate between the four screens using the bottom tab bar. On the Home screen, users can view an overview of the latest trends. The Current Trends screen allows users to browse a list of current trends, while the Match Trends screen might provide a way to compare trends. Finally, the Favorites screen enables users to manage their favorite trends, adding or removing them as needed.
The app provides a user-friendly interface for exploring and managing trends, making it easy for users to stay up-to-date with the latest information in their areas of interest.

3.14. Chrome Extension

Another inbound channel we used for more user interaction is using our “Intelligence Beyond Fashion” chrome extension to find the relevant products across different merchants.
Lets say you are in one amazon listing and you like the product but you are not satisfied with the product cost, you want to see similar products in other merchant websites as well. That is where our extension comes in picture. You just have to click on the start button of the extension and it will take the product title and product image and compare it against our product data and gives the similar products with price details.

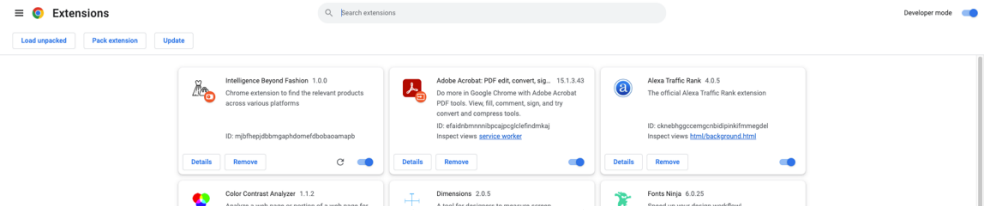


Figure 50 - Chrome extension deployment

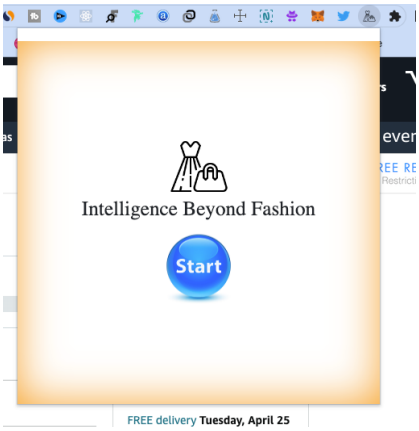


Figure 51 - Chrome extension

Refer to readme or runbook for more information on how to run the extension.

3.15. Flask Application

The complete backend model and APIs are exposed with the flask framework. There are three main API endpoints:

1. default or health – GET API – to check the health of the application
2. findrelativeimages – POST API – takes text and image input and provides the predicted list of text and images
3. lda – GET API – return current trends html visualization

Postman collection for the API reference is also included in the deliverable.

3.14. Deployment

Frontend deployment:

The ReactJS website is deployed in netlify and SSL configuration is also done.

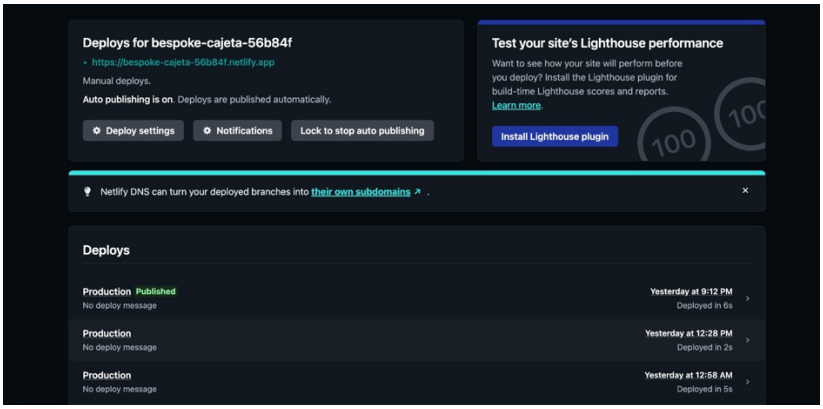


Figure 52 - Netlify deployment

Backend model/flask application deployment:

In case flask application, the following cloud resources were used:

1. Google cloud bucket – Stores the necessary model files

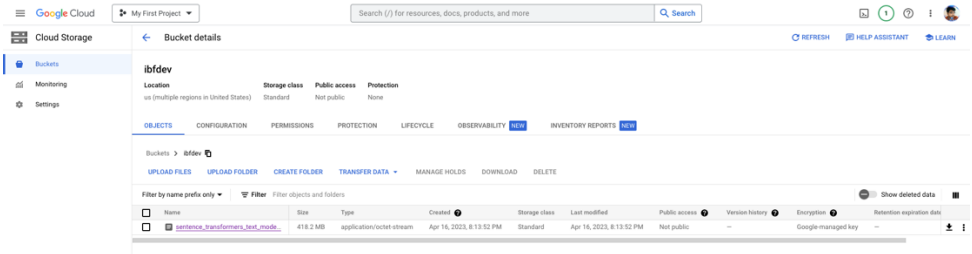


Figure 53 - Google cloud bucket

Credentials for this bucket is included in the flask application

2. AWS EC2 – Used to create instances with necessary storage for the application to run. Since we have used tensorflow, we needed c5.xlarge instance to run

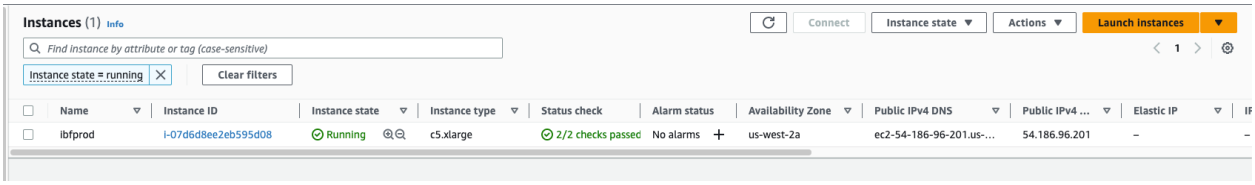


Figure 54 - EC2

Load balancers and security groups for this instances are also configured.

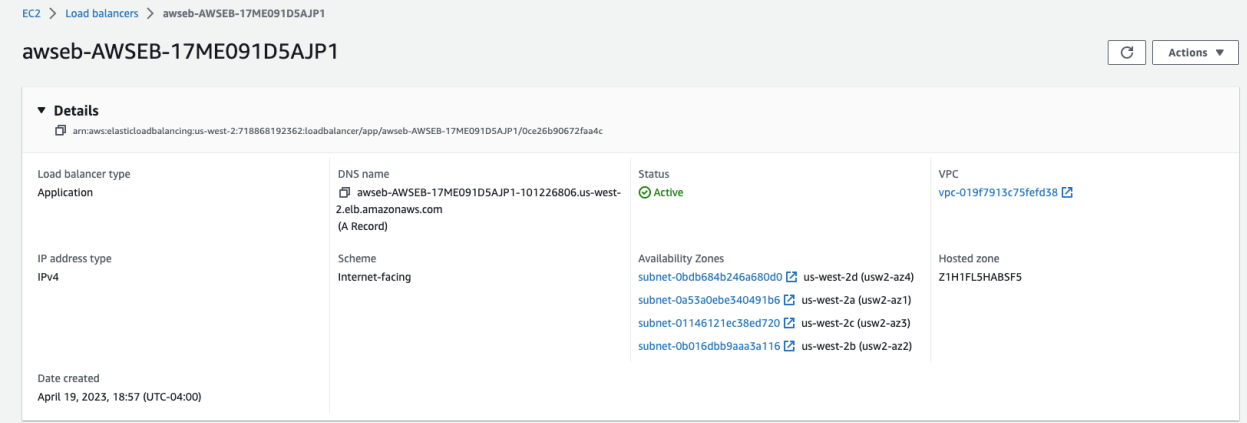


Figure 55 - Load balancers

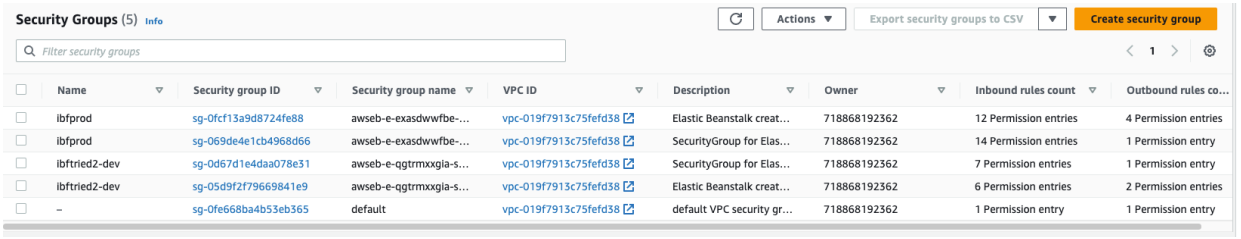


Figure 56 - Security groups

3. AWS Elastic Bean Stalk – Used to deploy the complete end to end load-balancing application which includes the creation ec2 instance, route configuration etc

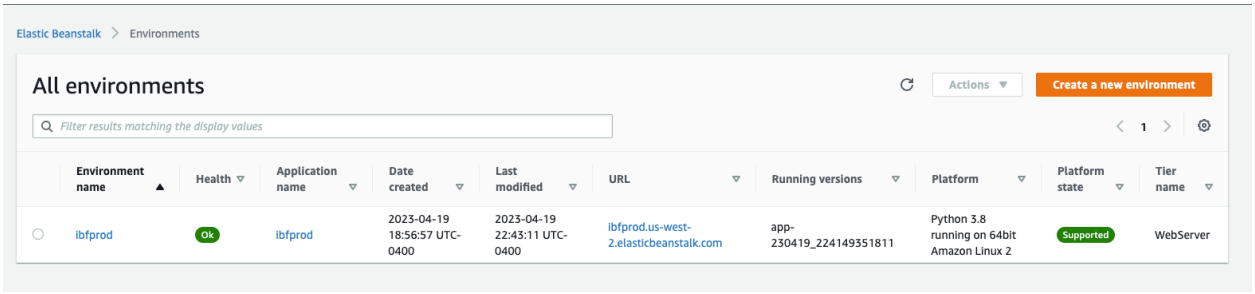


Figure 57 - Elastic bean stalk

4. AWS Load Balancer – Used for load balancing the instances for better scalability

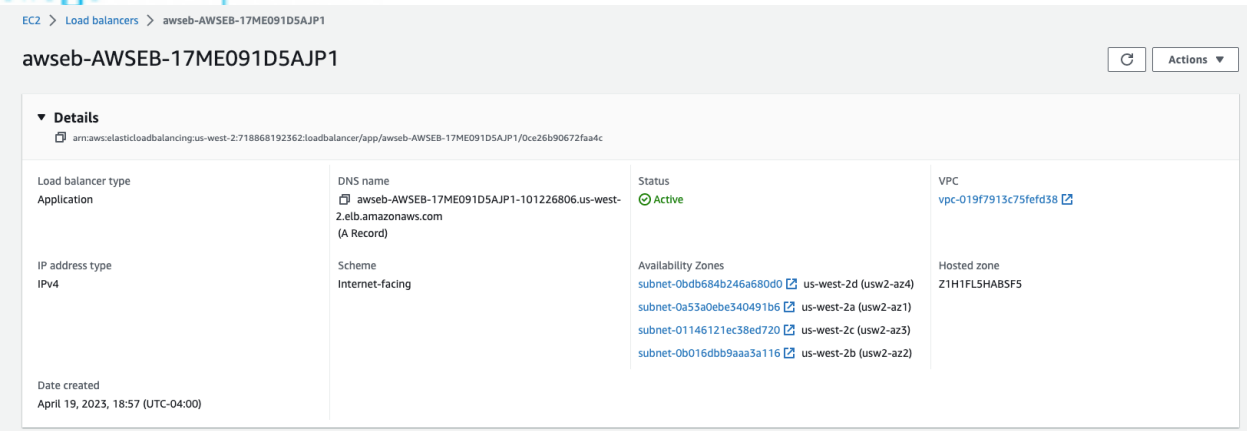


Figure 58 - Load balancer

5. AWS Certificate Manager – Used to create the necessary ssl for the domain name "fashiontrendcheck.com". This was necessary for the cross domain applications to work. Frontend which is hosted on netlify was not able to access the http port of the instance in AWS. So a domain name and SSL were necessary.

6. AWS Route 53 – Used for creating the domain name records and routing information

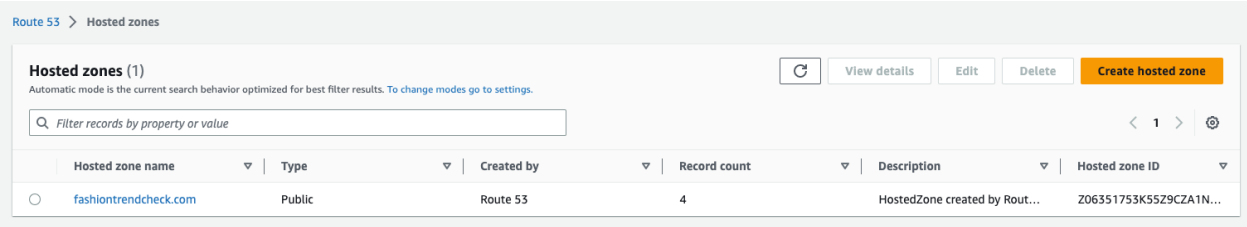


Figure 59 - Route53

7. AWS CloudFormation – Used for managing AWS resources in a secured manner

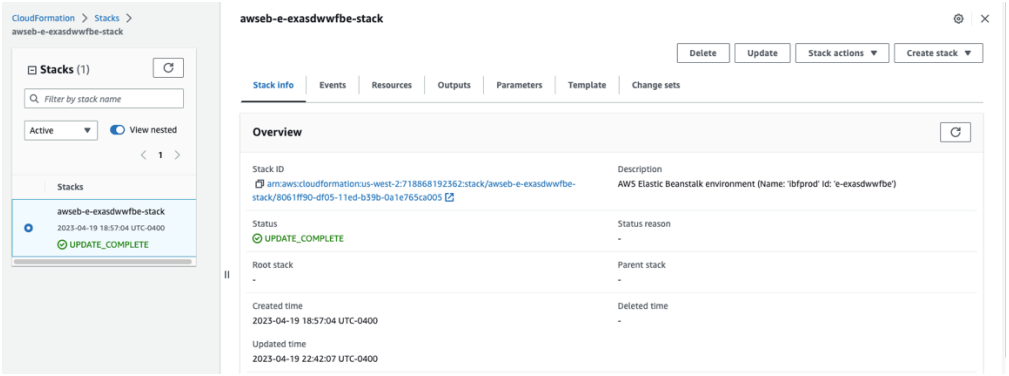


Figure 60 - Cloud formation

8. AWS CloudTrail – Used to setup the logs and risk management
9. AWS S3 – Used to store the product images and cloud trail logs

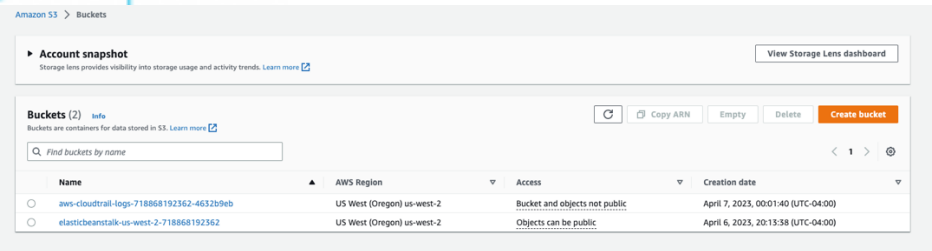


Figure 61 - Amazon S3

3.15. Bitbucket and JIRA

1. All the codes are uploaded directly to the Bitbucket repo.
   Project can be cloned using git clone https://nikeshhv@bitbucket.org/nikeshhv/intelligence-beyond-fashion.git
   However, due to the file size, not every model is added to the repository.
2. All the tasks are assigned and tracked using the project management tool "JIRA."
   Below is the velocity report for the complete project based on a total of 8 sprints. Each sprint runs every two weeks.

Figure 62 - JIRA velocity report

4. Results

In the case of TrendAnalysis,

Overall, the social media intelligence data obtained from Twitter analysis can be a valuable resource for small scale fashion industry analysts. By leveraging these insights, they can make informed decisions about product development, marketing strategies, and brand positioning, ultimately driving business growth and success.

In the case of TrendMatch,

As you can see, the Autoencoder decoder model (Type 2) performed better in finding similar images. However, it can be seen that when both text and images were combined, we were not able to calculate the efficiency because the data became unsupervised, and there was no source truth to compare it against.

The results of all the image models and their efficiency are given in section 3.

5. Conclusions and Future Work

Intelligence Beyond Fashion is a promising project that aims to provide users with a convenient and personalized shopping experience. The project addresses the issue of impulse buying and provides a solution by leveraging artificial intelligence to compare product details across different platforms. With the availability of multiple inbound channels, users can easily access the project from anywhere and at any time.

As a continuation, we want to increase the number of inbound channels and as well as improve the model accuracy. We have to go through a plugin based approach for inbound channels where many end users can interact with the system or application. Currently, we sometimes get products that are relevant. We need to find and fix those problems. However, due to resource complexity, we were not able to run and train the model again and again. We need to find a cost-effective method of model building. As we can see, models like Auto encoder decoder image model produced blurry images because we are trying to fit larger images into smaller vector space. So, we also would like to try out other models such as GAN, BERT etc.

In the future, the project can expand its scope beyond fashion and into other product categories. Additionally, the project can consider incorporating social media platforms to provide users with a more holistic view of the product, including user-generated content such as reviews and photos. The project can also explore the use of augmented reality to allow users to try on products virtually before making a purchase. Overall, the project has the potential to transform the online shopping experience, and future work can further enhance its capabilities.

6. References

Chaitanyanarava. (2020, December 17). Image similarity model. Medium. https://medium.com/analytics-vidhya/image-similarity-model-6b89a22e2f1a

Keras. (n.d.). Image classification from scratch. Retrieved April 17, 2023, from https://keras.io/examples/vision/image_classification_from_scratch/

Manas Narkar. (2020, November 12). Image classification with convolution neural networks (CNN) with Keras. Pythian Blog. Retrieved April 17, 2023, from https://blog.pythian.com/image-classification-with-convolution-neural-networks-cnn-with-keras/

Michał Oleszak. (2022, March 11). Autoencoders: From vanilla to variational. Towards Data Science. Retrieved April 17, 2023, from https://towardsdatascience.com/autoencoders-from-vanilla-to-variational-6f5bb5537e4a

AI Stack Exchange. Retrieved April 17, 2023, from https://ai.stackexchange.com/questions/8885/why-is-the-variational-auto-encoders-output-blurred-while-gans-output-is-crisp

What is the difference between CNN and a convolutional autoencoder? Quora. Retrieved April 17, 2023, from https://www.quora.com/What-is-the-difference-between-CNN-and-a-convolutional-autoencoder

Building a recommendation system using CNN v2. Kaggle. Retrieved April 17, 2023, from https://www.kaggle.com/code/marlesson/building-a-recommendation-system-using-cnn-v2

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet Allocation. Journal of Machine Learning Research, 3, 993-1022. https://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf

Lee, D. D., & Seung, H. S. (1999). Learning the Parts of Objects by Non-negative Matrix Factorization. Nature, 401, 788-791. https://www.nature.com/articles/44565

Chen, W., Wang, Y., & Yang, S. (2015). Efficient Parallel Algorithms for Non-negative Matrix Factorization and Latent Dirichlet Allocation on Multi-core and Multi-node Systems.

The Journal of Supercomputing, 71(3), 994-1013. https://link.springer.com/article/10.1007/s11227-014-1343-3

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems, 25, 1097-1105. https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556. https://arxiv.org/abs/1409.1556
Bengio, Y., Courville, A., & Vincent, P. (2013). Representation Learning: A Review and New Perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(8), 1798-1828. https://ieeexplore.ieee.org/document/6472238
Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P. A. (2008). Extracting and Composing Robust Features with Denoising Autoencoders. Proceedings of the 25th International Conference on Machine Learning, 1096-1103. https://dl.acm.org/doi/10.1145/1390156.1390294