**COURSE – INT248 – Advanced Machine**

**Learning**

|  |  |  |  |
| --- | --- | --- | --- |
| r. No. | Registration No | Name of Student | Roll No |
| 1 | **11805687** | **Ayush Soni** | **55** |
| 2 | **11805828** | **Ravi Shankar** | **56** |

**Git hub Link** https://github.com/raviuwe30/Building-Visual-Similarity-based-Recommendation

https://github.com/lonewolf292/Building-Visual-Similarity-based-Recommendation

**Submitted To Ankita Wadhawan**

Lovely Professional University

Jalandhar, Punjab, India.



…….

**Building Visual Similarity based Recommendation**

***Product recommendations*** can address such challenges very effectively by analysing the customer’s previous purchasing behaviour and current platform usage.

*Product recommendations* can help in:

* Converting the shoppers to customers
* Engaging the customers
* Boosting sales and revenue
* Delivering the most relevant content
* Maintaining the brand experience

Broadly speaking there are two kinds of recommendation approaches:

1. **Content-based recommendations**
2. **Collaborative filtering**

As the name suggests, the *content-based* method recommends based on the *additional content (metadata)* about the customers or products. For products, this *content* may be product title, description, images, category/subcategory, specification, etc.

So, this approach *recommends* the products by finding the most similar products to a given product based on the *content*.

In this post, we will implement a *content-based* recommendation system by utilising the *product images*. Basically, the goal is to recommend *product images*that are very similar to a recently bought/checked *product image*.

Therefore, this image-based recommendation will be helpful in *recommending* the most similar products to the customers based on their recent shopping behaviour/platform usage.

Let’s start implementing this using the Fashion product dataset. The dataset contains 2906 product images across four different *gender* categories (men, women, boys, and girls). It also contains various product features like title, category, subcategory, colour, gender, type, usage, etc.

kernel [here](https://www.kaggle.com/vikashrajluhaniwal/building-visual-similarity-based-recommendation)([*https://www.kaggle.com/vikashrajluhaniwal/building-visual-similarity-based-recommendation*](https://www.kaggle.com/vikashrajluhaniwal/building-visual-similarity-based-recommendation))  **Basic Data Analysis**

First few records from the dataset are as shown below.

Graphical user interface, text, application, email

Description automatically generated

**a.** **Basic statistics — Number of products, subcategories & gender**

print ("Total number of products: ", fashion\_df.shape[0])

print ("Total number of unique subcategories: ", fashion\_df["SubCategory"].nunique())

print ("Total number of unique gender types: ", fashion\_df["Gender"].nunique())

Total number of products: 2906

Total number of unique subcategories: 9

Total number of unique gender types: 4

dataset contains 2906 products of 9 different across 4 different gender types.

**b. Frequency of each gender**

fashion\_df["Gender"].value counts()

Men 811

Women 769

Boys 759

Girls 567

Name: Gender, dtype: int64

From the output, we can observe that most of the products belong to men category, then women, and so on.

**c. Distribution of products gender-wise**

**Chart, bar chart

Description automatically generated**

From the bar chart also, we can observe that men have the highest number of products. Similarly, the dataset is almost balanced.

## **2. Data Preparation**

Since cross-category recommendations are not preferred, for example, recommending girls’ products to a bachelor. So, let’s subset the data gender-wise into 4 different datagrams.

apparel\_boys = fashion\_df[fashion\_df["Gender"]=="Boys"]

apparel\_girls = fashion\_df[fashion\_df["Gender"]=="Girls"]

footwear\_men = fashion\_df[fashion\_df["Gender"]=="Men"]

footwear\_women = fashion\_df[fashion\_df["Gender"]=="Women"]

## **3. Feature Extraction using ResNet**

Generally, the product image contains a unique pattern along with its colour, shape, and edges.

Images with the same kind of such features are supposed to be similar. Therefore, extracting such features from the images will be very helpful in order to recommend the most similar products.

How to extract features from the images?

Computer vision techniques can be used to extract features from the images. Here, since we have limitations on data size, compute resources, and time, so let’s use the standard pre-trained models like ResNet to extract the features. Such pre-trained models are already fine-tuned and trained on a huge dataset (like ImageNet). This process is also known as transfer learning.

**ResNet**

ResNet is an abbreviated form of Residual Networks, first proposed by Kaiming He in 2015. Currently, it is perceived as a classical neural network for many computer vision tasks. In 2015, during the **ImageNet** Challenge, this model out-performed previous models like GoogleNet, VGGNet, and AlexNet.

A picture containing text, calculator

Description automatically generated

The architecture allows us to train an extremely deep and wide network with 152 layers successfully. In our implementation, we will use ResNet50 (a smaller version of ResNet152) to extract the features.

img\_width, img\_height = 224, 224

#top\_model\_weights\_path = 'resnet50\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5'

train\_data\_dir = "./Men/Images/"

nb\_train\_samples = 811

epochs = 50

batch\_size = 1

def extract\_features():

Itemcodes = []

datagen = ImageDataGenerator(rescale=1. / 255)

model = applications.ResNet50(include\_top=False, weights='imagenet')

generator = datagen.flow\_from\_directory(

train\_data\_dir,

target\_size=(img\_width, img\_height),

batch\_size=batch\_size,

class\_mode=None,

shuffle=False)

for i in generator.filenames:

Itemcodes.append(i[(i.find("/")+1):i.find(".")])

extracted\_features = model.predict\_generator(generator, nb\_train\_samples // batch\_size)

extracted\_features = extracted\_features.reshape((811, 100352))

np.save(open('./Men\_ResNet\_features.npy', 'wb'), extracted\_features)

np.save(open('./Men\_ResNet\_feature\_product\_ids.npy', 'wb'), np.array(Itemcodes))

a = datetime.now()

extract\_features()

print("Time taken in feature extraction", datetime.now()-a)

extract\_features() function extracts the features from the given images. As per the ResNet standard first, we resize the image to 224 x 224 and normalize them using **ImageDataGenerator** available in Keras. Finally, each image is represented as a 100352-dimensional feature vector.

To avoid run-time feature extraction after deployment, the extracted features are persisted in NumPy arrays. We maintain two arrays here for product Ids and extracted features respectively.

Similarly, this same feature extraction process is repeated for other product images gender-wise.

## **4. Computing the Euclidean distance and recommending similar products**

Distance is the most preferred measure to assess similarity among items/records. Minimum the distance, the higher the similarity, whereas, the maximum the distance, the lower the similarity.

There are various types of distances as per geometry like Euclidean distance, Cosine distance, Manhattan distance, etc. We will use Euclidean distance here to compute similarity.

Since we have already extracted the image features so the Euclidean distance can be easily computed using the pairwise\_distances() function form sklearn.metrics.

Once this distance is computed, we can easily recommend the products as per the ascending order of distance.

**a. Loading the extracted features**

|  |
| --- |
| extracted\_features = np.load('./Men\_ResNet\_features.npy') |
|  | Productids = np.load('./Men\_ResNet\_feature\_product\_ids.npy') |
|  |  |
|  | men = pd.read\_csv('./footwear\_men.csv') |
|  | df\_Productids = list(men['ProductId']) |
|  | Productids = list(Productids |

**b. Distance computation and Recommendation**

def get\_similar\_products\_cnn(product\_id, num\_results):

doc\_id = Productids.index(product\_id)

pairwise\_dist = pairwise\_distances(extracted\_features, extracted\_features[doc\_id].reshape(1,-1))

indices = np.argsort(pairwise\_dist.flatten())[0:num\_results]

pdists = np.sort(pairwise\_dist.flatten())[0:num\_results]

print("="\*20, "input product image", "="\*20)

ip\_row = men[['ImageURL','ProductTitle']].loc[men['ProductId']==int(Productids[indices[0]])]

#print(ip\_row.head())

for indx, row in ip\_row.iterrows():

display(Image(url=row['ImageURL'], width = 224, height = 224,embed=True))

print('Product Title: ', row['ProductTitle'])

print("\n","="\*20, "Recommended products", "="\*20)

for i in range(1,len(indices)):

rows = men[['ImageURL','ProductTitle']].loc[men['ProductId']==int(Productids[indices[i]])]

for indx, row in rows.iterrows():

display(Image(url=row['ImageURL'], width = 224, height = 224,embed=True))

print('Product Title: ', row['ProductTitle'])

print('Euclidean Distance from input image:', pdists[i])

get\_similar\_products\_cnn('13683', 5)

The above get\_similar\_products\_cnn() function recommends 5 most similar products to the queried product based on the extracted features. The function accepts two arguments — product id of recently bought/checked item and the number of products to be recommended.

The top 5 recommended products against the product id 13683 are as shown below.

A red high heel shoe

Description automatically generated with low confidenceA picture containing footwear, clothing, shoes, feet

Description automatically generated

Likewise, we can recommend products against the products from other gender types also. Let’s see the final deployment using **Streamlit**.

## **5. Deploying the Solution**

**Streamlit** is an interactive library/framework to build data apps & web applications and deploy machine learning workloads. The most important thing is it does not require any prior knowledge of web designing and development. Python knowledge is sufficient to interact with this as it is Python-compatible.

re, the below built-in functions are used to make an interactive deployment:

* st.text\_input() — takes dynamic input from the user
* st.write() — writes messages/arguments to the app
* st.title() — displays an image or list of images.

Like earlier, here the *get\_similar\_products\_cnn()* function *recommends* most similar products as per the arguments specified.

To execute this deployment script in the terminal type:

streamlit run recom\_deployment.py

**A pair of jeans

Description automatically generated with low confidence**

**A picture containing person, person, standing, trouser

Description automatically generated**