

Fashion Recommendation System

A PROJECT REPORT

Submitted by

**Varshini PJ
CB.SC.I5DAS18039**

*in partial fulfillment of the requirements for the award of the
degree of*

INTEGRATED MASTER OF SCIENCE

**IN
DATA SCIENCE**



AMRITA SCHOOL OF ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE 641112

JUNE 2022

AMRITA VISHWA VIDYAPEETHAM
AMRITA SCHOOL OF ENGINEERING, COIMBATORE, 641112



BONAFIDE CERTIFICATE

This is to certify that the project report entitled “**Fashion Recommendation System**” submitted by **CB.SC.I5DAS18039 Varshini PJ** in partial fulfillment of the requirements for the award of the **Degree of Integrated Master of Science in DATA SCIENCE** is a bonafide record of the work carried out under my guidance and supervision at Amrita School of Engineering, Coimbatore.

Class Advisor

Project Coordinator

Designation

Designation

Chairperson
Department of Mathematics
Dr. J. Ravichandran

The project was evaluated by us on:

Internal Examiner

External Examiner

AMRITA SCHOOL OF ENGINEERING
AMRITA VISHWA VIDYAPEETHAM

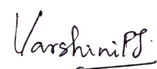
COIMBATORE - 641 112

DEPARTMENT OF MATHEMATICS

DECLARATION

I, **Varshini PJ** (Register Number-**CB.SC.I5DAS18039**), hereby declare that this dissertation entitled Fashion Recommendation System, is the record of the original work done by me. To the best of knowledge this work has not formed the basis for the award of any degree/diploma/associateship/fellowship/or a similar award to any candidate in any University.

Place: Coimbatore



Signature of the Student

Date:06/06/2022

COUNTERSIGNED

Dr. Prakash P

Project Advisors

Department of Mathematics

ACKNOWLEDGEMENTS

I would like to warmly acknowledge and express my deep sense of gratitude and indebtedness to Dr. Ravichandran J, Chairperson, Department of Mathematics, Amrita School of Engineering, for giving this wonderful opportunity to work on a great project like this.

I would like to thank my class advisor Dr. Prakash P, Department of Mathematics, and the faculty members of Department of Mathematics for allowing me to work for this Project.

My heartfelt thanks to all my friends for their invaluable co-operation and constant inspiration during my Project work.

I owe a special debt gratitude to my revered parents for their blessings and inspirations.

Coimbatore,
June 2022

Varshini PJ

Table of contents

ABSTRACT.....	1
1. INTRODUCTION.....	2
2. ABOUT THE DATASETS.....	2
2.3.1. Webscraping:.....	4
2.3.2. Bottlenecks faced:.....	6
3. MODEL BUILDING.....	7
3.1. About fastai:.....	7
3.2. Creating the Datablock:.....	8
3.3. Data Augmentation:.....	8
3.4. Choice of evaluating metric:.....	8
3.5. Choice of Loss function:.....	9
3.6. Challenges with the dataset:.....	9
3.7. Training the network with label smoothing:.....	10
3.8. Testing the model on the mapping data:.....	10
3.9. Color Prediction:.....	11
4. RECOMMENDATION SYSTEM AND UI CREATION.....	12
4.3.1. The preface:.....	15
4.3.2. Before uploading the image:.....	16
4.3.3. After uploading the image:.....	16
4.3.4. Recommendations:.....	17
5. CONCLUSION.....	18
6. SCOPE FOR IMPROVEMENT.....	18
7. REFERENCES.....	19

ABSTRACT

This project is aimed at helping black - owned small businesses in the category of Fashion by building a content-based Fashion Recommendation System, in which a user can get recommendations for an inspiration picture they upload, recommending products from various shopping sites. The recommendations are based on various attributes of the cloth in the uploaded picture, such as color, fabric, patterns, etc., The training dataset used is DeepFashion and the mapping dataset is the collection of products from the business sites, collected using webscraping. The recommendation system's outputs are displayed using a UI powered by streamlit, so that the recommended products can be linked and accessed by the user.

1. INTRODUCTION

This project is based on building a content-based Fashion Recommendation System. A content-based recommendation system takes user input and extracts data from it to suggest similar kind of objects (of interest of the user). For this case, attributes like color and patterns are considered.

2. ABOUT THE DATASETS

Building a recommendation system needs two datasets - the training and the mapping dataset. The training dataset is the DeepFashion dataset (which is available only for non-commercial purposes). The mapping dataset is created using Webscraping using BeautifulSoup in Python.

2.1. DeepFashion:

The DeepFashion dataset has the following properties:

- ◆ DeepFashion contains over **800,000** diverse fashion images ranging from well-posed shop images to unconstrained consumer photos, constituting the largest visual fashion analysis database.
- ◆ DeepFashion is annotated with rich information of clothing items. Each image in this dataset is labeled with **50** categories, **1,000** descriptive attributes, bounding box and clothing landmarks.
- ◆ DeepFashion contains over **300,000** cross-pose/cross-domain image pairs.

There are 4 types of benchmark datasets under DeepFashion. This project uses “**Category and Attribute Prediction Benchmark**” dataset under DeepFashion. The dataset is liable only for non-commercial purposes with the author’s approval.

2.2. Category and Attribute Prediction Benchmark dataset:

This benchmark dataset was created by Ziwei Liu et al., which involves collection and annotation of various clothing images. It is a 3gb dataset which contains low resolution images (high resolution images are a bottleneck).

It has 38 classes - Anorak, Blazer, Blouse, Bomber, Button-Down, Caftan, Capris, Cardigan, Chinos, Coat, Coverup, Culottes, Cutoffs, Dress, Flannel, Gauchos, Halter, Henley, Hoodie, Jacket, Jeans, Jeggings, Jersey, Jodhpurs, Joggers, jumpsuit, Kaftan, Kimono, Leggings, Onesie, Parka, Peacoat, Poncho, Robe, Romper, Sarong, Shorts, Skirt, Sweater, Sweatpants, Sweatshorts, Tank, Tee, Top, Trunks, Turtleneck.

This dataset contains 289,222 diverse clothes images labeled with 1000 different attributes. From which I used 98 attributes that correspond to style, fabric, season, and type of the pattern such as abstract-print, animal, baroque, basic, beach, bird-print, boho, botanical-print, camouflage, cargo, chic, chiffon, etc.,

Glimpse of the downloaded dataset:

Name	Date modified	Type	Size
2-in-1_Space_Dye_Athletic_Tank	24-04-2022 20:52	File folder	
25_Mesh-Paneled_Jersey_Dress	24-04-2022 20:52	File folder	
36_Plaid_Shirt_Dress	24-04-2022 20:52	File folder	
1981_Graphic_Ringer_Tee	24-04-2022 20:52	File folder	
Above_Average_Linen_Tee	24-04-2022 20:52	File folder	
Abstract_Animal_Print_Dress	24-04-2022 20:52	File folder	
Abstract_Arrow_Flounce_Romper	24-04-2022 20:52	File folder	
Abstract_Asymmetrical_Hem_Top	24-04-2022 20:52	File folder	

Name	Date modified	Type	Size
attribute-classes.txt	06-04-2022 09:28	Text Document	1 KB
classes.txt	06-04-2022 09:28	Text Document	1 KB
multilabel-test.csv	06-04-2022 09:28	CSV File	1,272 KB
multilabel-train.csv	06-04-2022 09:28	CSV File	8,455 KB
test_labels.csv	06-04-2022 09:28	CSV File	2,096 KB
train_labels.csv	06-04-2022 09:28	CSV File	13,049 KB

2.3. Mapping Dataset:

The mapping dataset has the data which a recommender uses to give recommendations. In this project, the mapping dataset comprises of data from several black-owned small businesses such as Ofuure, Joile Noire, Tove Studio etc., The data is collected using Webscraping using BeautifulSoup, code in Python.

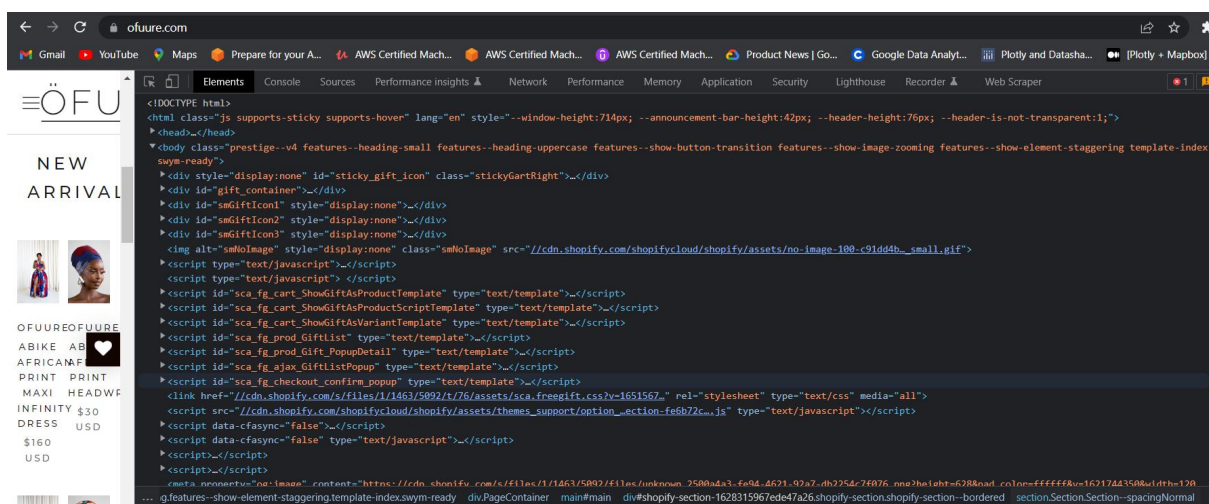
2.3.1. Webscraping:

Web scraping is an automatic method to obtain large amounts of data from websites. Most of this data is unstructured data in an HTML format which is then converted into structured data in a spreadsheet or a database so that it can be used in various applications. There are many different ways to perform web scraping to obtain data from websites. Web Scrapers can be divided on the basis of many different criteria, including Self-built or Pre-built Web Scrapers, Browser extension or Software Web Scrapers, and Cloud or Local Web Scrapers. This project uses a **Self-built webscraper** using BeautifulSoup.

2.3.1.1. Web Scraping demonstrated for a website:

Website: <https://www.ofuure.com/>

We can find the HTML structure of a website by right clicking on any place on a site and choosing “Inspect”. Inspect for this website looks like this:



Inside this nested structure of HTML tags, we need to find where our needful information lies and feed that into our scraping code.

The information that we need to scrape are: Product Links, Image links, Product name, and Price.

The HTML tags inside which the information is kept, changes from website to website depending on version and other factors.

2.3.1.2. Scraping the product links:

Ofuure has 45 pages containing all products. All the product links are kept inside the “div” tag and “ProductItem” class.

```
product_links=[]
for x in range(1,46):
    k = requests.get(f'https://www.ofuure.com/collections/all-products?page={x}')
    soup = BeautifulSoup(k.content, "lxml")
    productlist = soup.find_all("div",class_="ProductItem")
    #print(productlist)

    for item in productlist:
        for link in item.find_all("a",href=True):
            product_links.append(baseurl + link['href'])
    productlinks = list(dict.fromkeys(product_links))
print(len(productlinks))
```

898

There were 898 products in total. The product list/ product names are taken from the “a” tag and “href” class.

2.3.1.3. Scraping Image links and prices:

The image links are kept inside a tag called “property” and under the class “og:image:secure_url”.

```
descriptionlist,imagelink,price =[],[],[]
for link in productlinks:
    r = requests.get(link,headers=headers)
    soup = BeautifulSoup(r.content, "lxml")

    for tag in soup.find_all("meta"):
        if tag.get("property", None) == "og:image:secure_url":
            imagelink.append(tag.get("content", None))
        if tag.get("property", None) == "product:price:amount":
            price.append(tag.get("content", None))
        if tag.get("property", None) == "og:description":
            t=tag.get("content", None)
            descriptionlist.append(t.strip())
descriptionlist = " ".join(descriptionlist)
```

All the images are of the form “https://cdn.shopify.com/s/files/”. Its common for all the websites to upload their images to shopify and link them in their websites.

All the scraped data is saved in a form of a dataframe and made into a .csv file.

After scraping the basics, we've to all the images from the site.

2.3.1.4. Scraping Product Images:

The image links scraped from the above activity can be used to scrape the images from the shopify site.

```
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

prodlinks = data['Image Link']
for i in range(len(prodlinks)):
    urllib.request.urlretrieve(prodlinks[i], f"images/{i}.jpg")
```

All the images were scraped and stored in separate folders with respect to brand names.

📁 Jolie Noire	02-04-2022 10:43	File folder
📁 Naclo Apparel	08-05-2022 00:15	File folder
📁 OFUURE	18-03-2022 01:31	File folder
📁 TOVE STUDIO	02-04-2022 10:43	File folder
📁 Zanyus	02-04-2022 12:22	File folder

2.3.2. Bottlenecks faced:

There were several bottlenecks faced during scraping the images from the shopping sites. Web scraping involved inspection of the website, understanding the HTML structure and tags was a tedious process. Varying versions of HTML caused many changes in scraping codes. Separation of 'cloth-alone' pictures from 'model-wearing-cloth' pictures during scraping was a bottleneck. Some websites like Naclo Apparel and Zanyus had a dynamic structure where image sizes varied between {180w, 360w, 720w, ...}. Image links varied accordingly and the switch case made it tough to scrape. The fact that not to put too much strain on a website by continually scraping, also was hard. It might increase traffic/load on the website which may cause in crashing.

The next step is model building, followed by training the model on images from the DeepFashion dataset.

3. MODEL BUILDING

3.1. About fastai:

We use a very useful python package for the model implementation part - fastai. fastai is a deep learning library which provides practitioners with high-level components that can quickly and easily provide state-of-the-art results in standard deep learning domains, and provides researchers with low-level components that can be mixed and matched to build new approaches. It aims to do both things without substantial compromises in ease of use, flexibility, or performance. This is possible thanks to a carefully layered architecture, which expresses common underlying patterns of many deep learning and data processing techniques in terms of decoupled abstractions. These abstractions can be expressed concisely and clearly by leveraging the dynamism of the underlying Python language and the flexibility of the PyTorch library.

Training data are stored in multilabel-train.csv . It contains the following data:

	image_name	labels	is_valid
0	img\Sheer_Pleated-Front_Blouse\img_00000002.jpg	pleated,sheer	0.0
1	img\Sheer_Pleated-Front_Blouse\img_00000003.jpg	chic,pleated	1.0
2	img\Sheer_Pleated-Front_Blouse\img_00000006.jpg	pleated	1.0
3	img\Sheer_Pleated-Front_Blouse\img_00000009.jpg	sheer,sophisticated	0.0
4	img\Sheer_Pleated-Front_Blouse\img_00000010.jpg	relaxed,sheer,woven	0.0

The dataset contains the location of the images, labels stored as a string object, and whether the image belongs to a validation set.

```
def get_x(r): return PATH+r['image_name']
def get_y(r): return r['labels'].split(',')

def splitter(df):
    train = df.index[df['is_valid']==0].tolist()
    valid = df.index[df['is_valid']==1].tolist()
    return train,valid
```

To load the images, we use a *DataBlock*, that expects as input the full path to the image as an independent variable and the list of strings as the labels. For that, we create two helper functions to parse the input format -get_x and the get_y.The splitter function will create train and validation sets based on the *is_valid* column.

3.2. Creating the Datablock:

```
dblock = DataBlock(blocks=(ImageBlock, MultiCategoryBlock),
                    splitter=splitter,
                    get_x=get_x,
                    get_y=get_y,
                    item_tfms=RandomResizedCrop(224, min_scale=0.8),
                    batch_tfms=aug_transforms())

dls = dblock.dataloaders(train_df, num_workers=0)
```

We specify the type of the independent variable as an ImageBlock, and the label as MultiCategoryBlock. MultiCategoryBlock expects labels to be stored as a list of strings.

3.3. Data Augmentation:

RandomResizedCrop randomly crops a part of the image. Each epoch it crops a different part of the image. This is used in the latter part of the code to zoom into the pictures if any model-cloth picture is involved.

aug_transforms - creates a list of augmentations such as horizontal flip, rotation, zoom, warp, and lighting transforms.

3.4. Choice of evaluating metric:

The list of labels that we passed to the DataBlock was transformed to the sparse One-Hot encoded vector. The sparsity will affect our choice of the evaluation metric. The default choice for the evaluation metric in fastai is *accuracy_multi*. If we create a tensor of zeroes and send it to *accuracy_multi* together with the target values we will already achieve 98% accuracy. This means that if our model would not predict any label, it would already be 98% accurate. Therefore we need another evaluation metric, that will reflect the realistic learning progress and performance of our model.

3.4.1. Fbeta score:

FBeta is a generalization of **F-score**. While **F-score** is defined as the harmonic mean of precision and recall and gives each the same weight, **FBeta** adds a configuration parameter called beta. The beta parameter determines the weight of recall in the combined score. $\beta < 1$ lends more weight to precision, while $\beta > 1$ favors recall ($\beta \rightarrow 0$ considers only precision, $\beta \rightarrow +\infty$ only recall).

In case of no prediction, it will give us a 0 FBeta score.

3.5. Choice of Loss function:

Looking at the example of training labels. It is represented as a One-Hot encoded vector. This type of encoding is also known as a **hard encoding**.

[illegible]

When we use hard encoded labels, the predicted probabilities for correct labels will be extremely large and for the incorrect classes extremely low. The model will classify every training example correctly with the confidence of almost 1. This might lead to multiple problems; the first problem is overfitting. The other two problems are related to the nature of labels in our dataset.

3.6. Challenges with the dataset:

The clothes attribute labels are semantically very close. For example: 'botanical-print' and 'floral', or 'faux-leather' and 'leather' are very hard to distinguish even for a human. When we train a network with semantically similar labels encoded as One-Hot, the difference between correct and incorrect classes is extremely high, even though they might be very close semantically. The confidence with which model will predict the labels is not limited and might vary across classes, which makes the separation of a correct class more challenging.

3.6.1. How to overcome?

Labels smoothing prevents the network from becoming over-confident and it will less likely overfit. Moreover, it makes the differences between the correct and incorrect classes constant by introducing a smoothing parameter α . As a result, each incorrect class label will be equidistant from the correct ones which make it easier to separate semantically similar classes. We will implement the label smoothing inside the loss function and use it to train our model.

In fastai the default choice of the loss function for a multi-label classification problem is a BCEWithLogitsLossFlat, which is a Binary Cross Entropy with Logits (in fact a sigmoid function, which is an inverse of the logit function and maps arbitrary real values of Cross-Entropy back to the $[0, 1]$ range which is what we usually want for the targets encoded between 0 and 1).

```

class LabelSmoothingBCEWithLogitsLossFlat(BCEWithLogitsLossFlat):
    def __init__(self, eps:float=0.1, **kwargs):
        self.eps = eps
        super().__init__(thresh=0.2, **kwargs)

    def __call__(self, inp, targ, **kwargs):
        targ_smooth = targ.float() * (1. - self.eps) + 0.5 * self.eps
        return super().__call__(inp, targ_smooth, **kwargs)

    def __repr__(self):
        return "FlattenedLoss of LabelSmoothingBCEWithLogits()"

```

We'll be using a resnet architecture with 34*34 embeddings.

3.7. Training the network with label smoothing:

```

learn = cnn_learner(dls, resnet34, loss_func=LabelSmoothingBCEWithLogitsLossFlat(),
                    metrics=metrics, opt_func=opt_func).to_fp16()

```

```

learn.fine_tune(2)
learn.save('atr-recognition-stage-1-resnet34')

```

Training the model and fine tuning it with the Label smoothing function for 2 iterations. This process is computationally intensive and took me around 6 hours to complete training.

epoch	train_loss	valid_loss	fbeta_score	accuracy_multi	time
0	0.227698	0.225592	0.406759	0.983390	5:54:49
1	0.226540	0.224695	0.427865	0.983679	6:02:50

```

Path('models/atr-recognition-stage-1-resnet34.pth')

```

```

learn.export('atr-recognition-stage-1-resnet34.pkl')

```

The model is saved as a .pkl file and can be used the further development without the need for re-training.

3.8. Testing the model on the mapping data:

After importing all the dependencies of the saved model, the model is all set to be tested on the mapping data. Path of the scraped images was engineered.

The predictions of the model were collected and saved into the respective dataframe of the brand.

prediction = predict_attribute(model, image_path)

3.9. Color Prediction:

After predicting the attributes of the mapping dataset, we additionally also identify the color of the dresses. This is to help the recommendation system to not only suggest dresses based on the attributes but also the color.

In order to perform the task of detecting the color of the dresses – the following dependencies used are the K-Means algorithm for finding the most prominent colors and the modules called colormap, webcolors for the purpose of converting them from hex colors to color names.

We'll be finding 2 colors - actual and the closest color of the cloth that is present in the mapping data.

```
def closest_colour(requested_colour):
    min_colours = {}
    for key, name in webcolors.CSS3_HEX_TO_NAMES.items():
        r_c, g_c, b_c = webcolors.hex_to_rgb(key)
        rd = (r_c - requested_colour[0]) ** 2
        gd = (g_c - requested_colour[1]) ** 2
        bd = (b_c - requested_colour[2]) ** 2
        min_colours[(rd + gd + bd)] = name
    return min_colours[min(min_colours.keys())]

def get_colour_name(requested_colour):
    try:
        closest_name = actual_name = webcolors.rgb_to_name(requested_colour)
    except ValueError:
        closest_name = closest_colour(requested_colour)
        actual_name = None
    return actual_name, closest_name
```

The above code makes use of KMeans algorithm to take the centroid of the clusters to find the actual and the closest colors. KMeans algorithm returns the centroid of the clusters as a 3-value pair of RGB values. Since, for the cluster centroid the average of the RGB values is taken. The value we receive won't always have an exact RGB to name conversion. Hence, we find the closest color in order to map it to the closest RGB value which has a color name.


```

for i in file_names:
    image_path = PATH + "/" + i
    img = cv2.imread(image_path) # Since, OpenCV opens image in BGR mode, and we require RGB for correct image
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB) # Also, we use cvtColor instead of imread
    plt.imshow(img)
    plt.show()
    height, width, dim = img.shape
    # Take the center of the image
    #img = img[(height//3):(2*height//3), (width//3):(2*width//3), :]
    height, width, dim = img.shape
    img_vec = np.reshape(img, [height * width, dim] )
    plt.imshow(img)
    plt.show()

    # Use k-means to cluster the pixels together
    kmeans = KMeans(n_clusters=2)
    kmeans.fit(img_vec)
    # Identify the colors
    unique_l, counts_l = np.unique(kmeans.labels_, return_counts=True)
    #print(unique_l,counts_l)
    sort_ix = np.argsort(counts_l)
    sort_ix = sort_ix[::-1]
    for cluster_center in kmeans.cluster_centers_[sort_ix]:
        #print(cluster_center)
        r = int(cluster_center[0])
        g = int(cluster_center[1])
        b = int(cluster_center[2])
        actual_color,closest_color = get_colour_name((r,g,b))
        if len(temp)==1:
            temp.append(closest_color)
        else:
            temp = []
            temp.append(closest_color)
    color.append(temp)

```

The color attributes are also saved into the dataframe. Similarly, it is done for all the cloths in the various brands and saved as a single dataframe.

4. RECOMMENDATION SYSTEM AND UI CREATION

Both the processes are done unanimously to achieve the end product which is a UI powered Fashion Recommendation System.

The UI creation involves the usage of a tool/package called “streamlit”.

4.1. About Streamlit: Streamlit is an open-source app framework in Python language. It helps us create web apps for data science and machine learning in a short time. It is compatible with major Python libraries such as scikit-learn, Keras, PyTorch, SymPy(latex), NumPy, pandas, Matplotlib etc.

4.2. Recommendation System:

The recommendation system works on the user’s uploaded image. It first tries to predict the cloth attributes and its corresponding color for the user’s uploaded image of choice. The next step, is to discover the similar looking dresses

available to display to the user. Next, we access the database that we have created for all the different brands and tries to compare the attributes and color. It then fetches all the dresses that it suggests as similar to the user's uploaded image, along with the other details about the dress such as the brand name, product name, price, and product link. It also fetches the image of the dress from the corresponding storage location to display to the user.

The recommendation system chooses the similar looking dresses, based on a scoring criterion that I have created for this app. The higher the score, the more similar the dress is to the user's expectations. We display the dresses in the same order of highest to lowest scored dresses.

The code for the scoring criterion is as mentioned below,

```
for i in range(0, len(data)):

    score = 0

    color_check = data['color'][i]

    type_check = data['prediction'][i]

    for type in prediction:

        if type in type_check:

            score += 1

    if color in color_check:

        score += 1

    score_color.append(score)

    selected_list_color.append(data['Image Path'][i])

    brand_color.append(data['Company'][i])

    product_title_color.append(data['Product Name'][i])

    cost_color.append(data['Price'][i])

    urls_color.append(data['Product Link'][i])

    # Sort based on the score calculation for most similar first

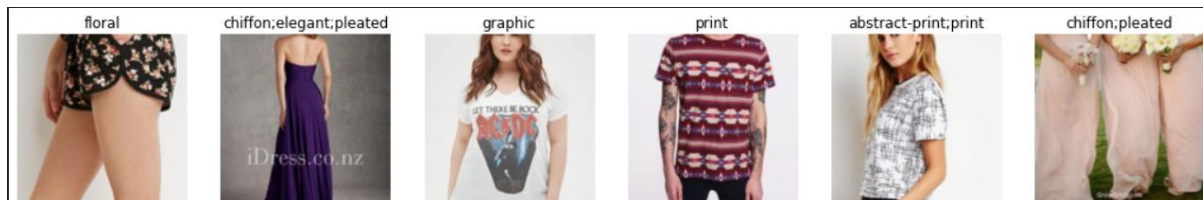
    score_color, selected_list_color, brand_color, product_title_color, cost_color, urls_color =
sort_together(

        [score_color, selected_list_color, brand_color, product_title_color, cost_color, urls_color],
reverse=True)
```

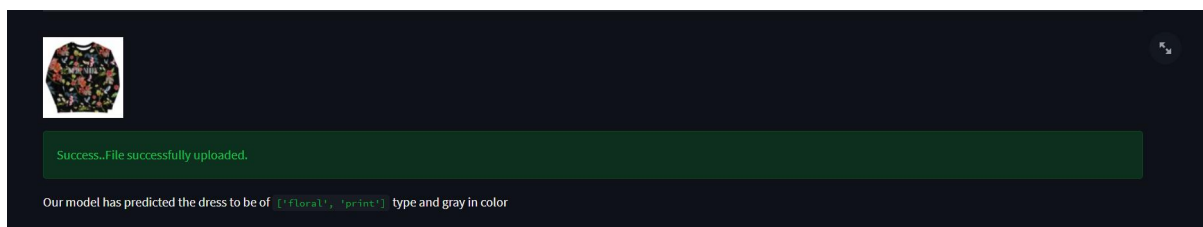
Since our model is a multi-class prediction model, it can sometimes predict multiple classes for a single dress. Similarly, for the predictions made on our mapping dataset are also multi-class. Hence, we need to compare each class separately and depending on how many classes matches we need to give a

corresponding score (i.e) if one class matches with the user's input, we give it a score of 1 and if 2 classes match, we give it a score of 2. Similarly, for the color for every color match a score of 1 is added. By adding all these scores together, we get the final score for how similar our dress is with the user's uploaded dress.

The below picture shows that an image in the dataset being annotated to multiple classes:



Our model's multiclass prediction:



The following image has been classified as: “floral” and “print”



4.3. UI part:

4.3.1. The preface:

Our recommendation system works on the user's uploaded image. Hence, the users should be able to upload their image files easily and confidently. The below is the streamlit functionality, which incorporates the title header, display message and the upload files button.

It uploads the user's image, and notes down the file details such as file name, file type and file size for record purposes. The user is also provided the option to view these details about the file they uploaded if needed.

```
st. set_page_config(layout="wide")

st.title('Fashion Recommendation System')

st.markdown("-----")

# Getting the user's input image
image_file = st.file_uploader("Upload Image", type=['png', 'jpeg', 'jpg'])

if image_file is not None:

    file_details = {"Filename": image_file.name,

                    "FileType": image_file.type, "FileSize": image_file.size}

    file_button = st.button("Show File Details",

                            help="Click here to view the file details")

    if file_button:

        st.write(file_details)

    st.markdown("-----")

    img = load_image(image_file)

    st.image(img, width=100)

    # Save the file in a temp location for the model to predict using

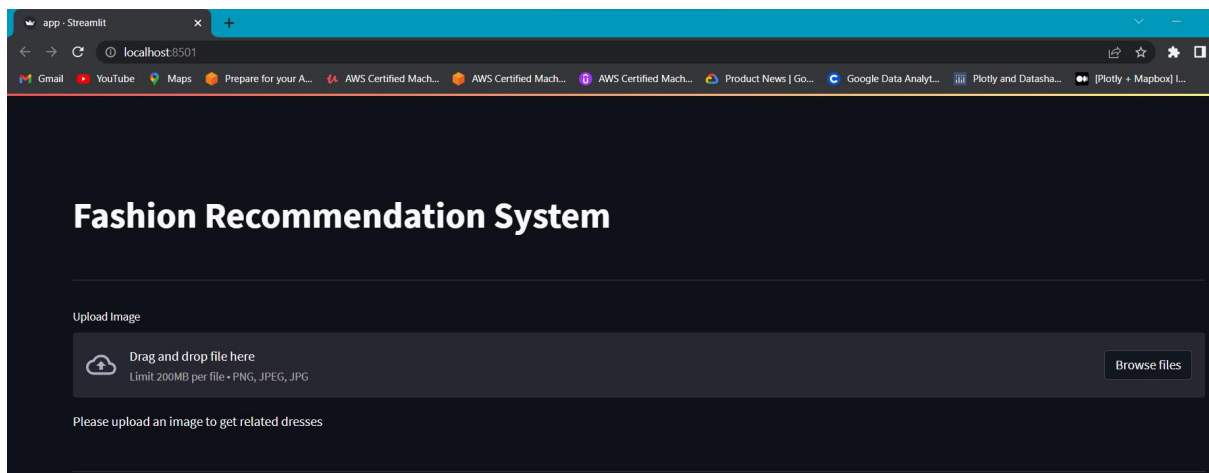
    img_name = 'temp'

    with open(os.path.join("tempDir", img_name), "wb") as f:

        f.write(image_file.getbuffer())

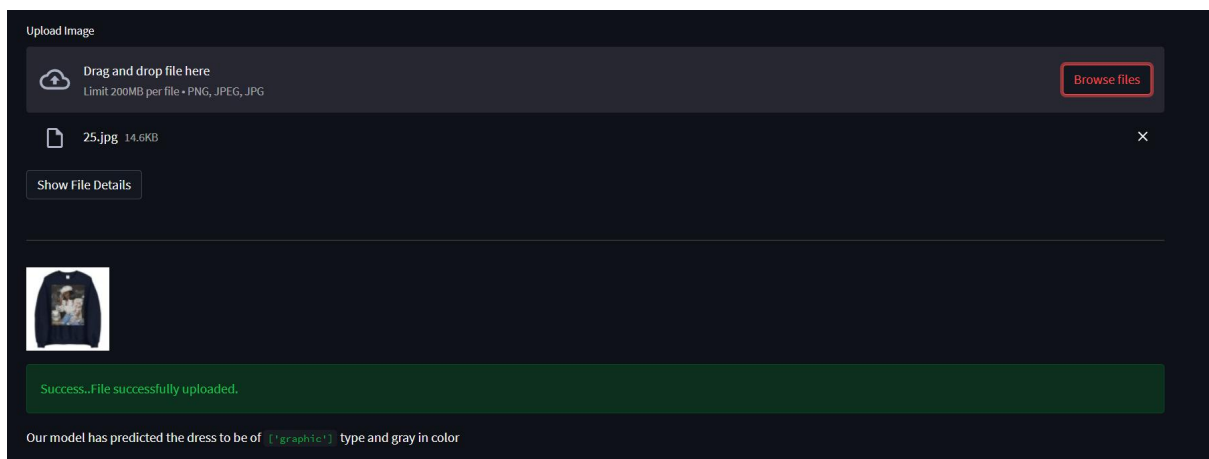
    st.success("Success..File successfully uploaded.")
```

4.3.2. Before uploading the image:



The user can either drag and drop or can browse through their machine's directory. These interfaces are powered by streamlit's functionalities. The users are provided with a rich variety of image type options to upload such as png, jpeg and jpg.

4.3.3. After uploading the image:



Once the user successfully uploads an image, a small display of the image uploaded is shown to the user along with a success message to let the user know that the file upload was a success.



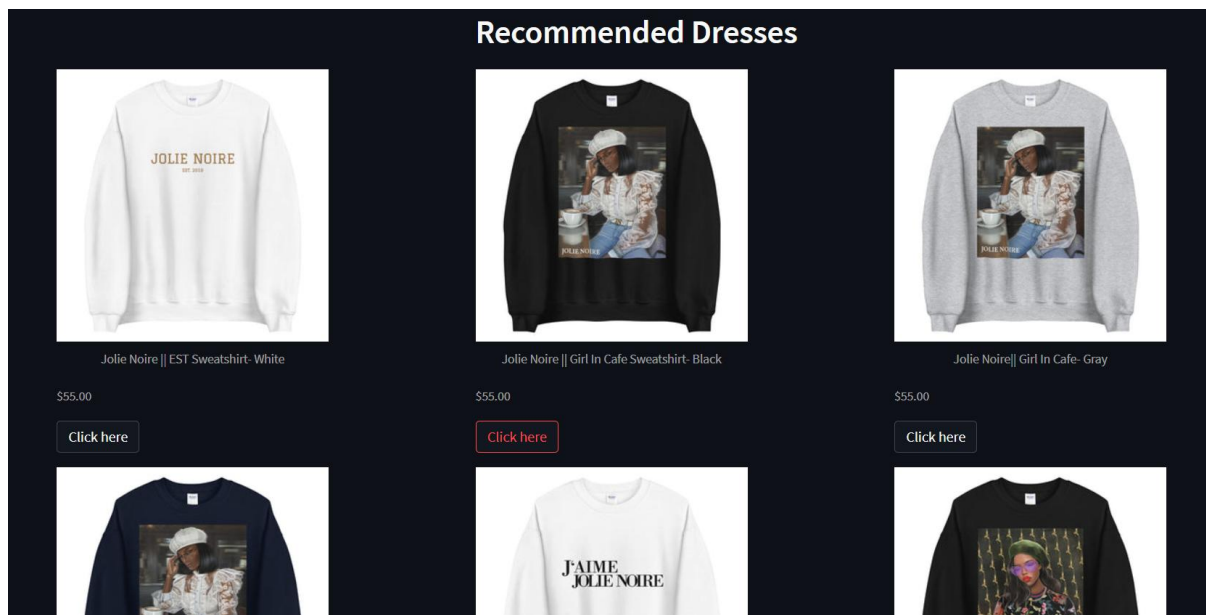
This is the uploaded image, the model predicted it to be a “graphic” type and “gray” color.

4.3.4. Recommendations:

The user uploaded the above image, and our model predicted the type and color as mentioned above. Now using the scoring criterion discussed above, the recommendation system has identified the most similar dresses. The dresses are displayed to the user, based on their scores. In the UI, the most to least similar dresses will be retrieved and displayed in 3 columns.

The user can view the similar dresses, the brand, the price, product name and also the option to visit the actual website to purchase or view other details about using the “Click here” button which adds value to the businesses.

When we look at the recommendations made by our model, we can clearly see that our model performs quite good with finding the attributes, color and recommending similar dresses for the user.



5. CONCLUSION

We have successfully built a high-end working fashion recommendation system web-based application, which is able to get input images from the user and predict the type of cloth, color and make recommendations suited for the user's requirement from the existing database consisting of multiple brands. This application has a lot of scope in the real-world situations mainly for fashion-based shopping apps which can use such recommendation systems to help user's find their desired dresses in a much more convenient way and also help the smaller known brands to reach out to the potential customers who are looking for a similar kind of fashion. An inspiration to the project idea of building a Fashion recommendation system was the idea to build a content-based recommendation system to support black owned small businesses, being an ally to the Black Lives Matter Movement. All the brands data that has been scraped and used in the recommendation system are from some of the black owned small businesses.

6. SCOPE FOR IMPROVEMENT

The training dataset used for the prediction model actually had 2 types available – low resolution and high resolution. The dataset used was the low-resolution dataset due to the lacking of computational power. Even using the low-resolution dataset, we could only perform training for 2 epochs which took us around 12 hours using a good laptop of specs i7 CPU and 6GB GPU. As we could see even with just such little training and using the low-res dataset, we

were able to achieve such good predictions on the dresses type and recommendations for similar dresses. If we could use better computing power to train the model, we could achieve higher accuracy predictions on the type of dress and recommend on a better scale.

7. REFERENCES

Paper for DeepFashion :

https://openaccess.thecvf.com/content_cvpr_2016/papers/Liu_DeepFashion_Powering_Robust_CVPR_2016_paper.pdf

ResNet34 :

<https://models.roboflow.com/classification/resnet34#:~:text=Resnet34%20is%20a%2034%20layer,images%20across%20200%20different%20classes.>

Streamlit documentations:

<https://docs.streamlit.io/>

Cosmopolitan 100 black owned businesses:

<https://www.cosmopolitan.com/style-beauty/fashion/g32733776/black-owned-clothing-brands/>

Inspiration:

<https://github.com/codingbunnie/WISH-by-NANA-FashionRecommendationEngine>