AIWIR PROJECT

FASHION RECOMMENDER SYSTEM

TEAM MEMBERS [TEAM 17] :

PRIYA MOHATA	PES2UG19CS301
PRIYANSH GAURAV	PES2UG19CS302
R SHARMILA	PES2UG19CS309
RITIK	PES2UG19CS332

TABLE OF CONTENTS

01

INTRODUCTION

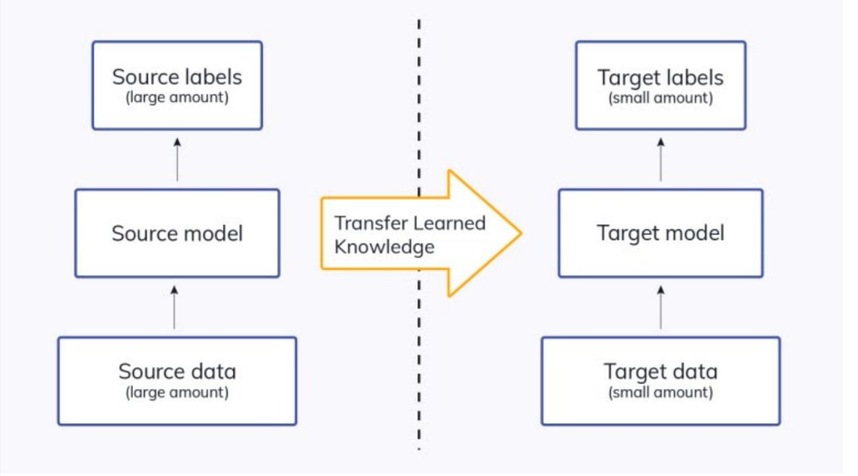
About Fashion Recommender Systems :

In recent years, the textile and fashion industries have witnessed an enormous amount of growth in fast fashion. On e-commerce platforms, where numerous choices are available, an efficient recommendation system is required to sort, order, and efficiently convey relevant product content or information to users. Image-based fashion recommendation systems (FRSs) have attracted a huge amount of attention from fast fashion retailers as they provide a personalized shopping experience to consumers. With the technological advancements, this branch of artificial intelligence exhibits a tremendous amount of potential in image processing, parsing, classification, and segmentation.

An effective recommendation system is a crucial tool for successfully conducting an e-commerce business. Fashion recommendation systems (FRSs) generally provide specific recommendations to the consumer based on their browsing and previous purchase history. Social-network-based FRSs consider the user's social circle, fashion product attributes, image parsing, fashion trends, and consistency in fashion styles as important factors since they impact upon the user's purchasing decisions. FRSs have the ability to reduce transaction costs for consumers and increase revenue for retailers.

Transfer Learning :

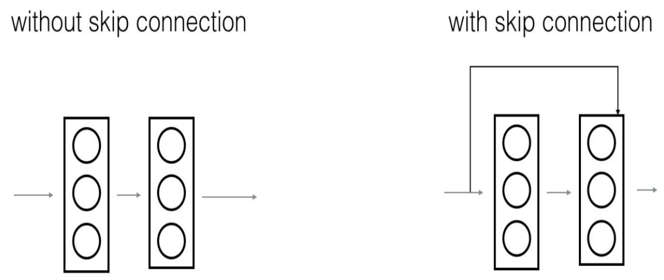
- Transfer Learning is a machine learning method where we reuse a pre-trained model as the starting point for a model on a new task.
- By applying transfer learning to a new task, one can achieve significantly higher performance than training with only a small amount of data.
- To put it simply—a model trained on one task is repurposed on a second, related task as an optimization that allows rapid progress when modeling the second task.
- ImageNet, AlexNet, and Inception are typical examples of models that have the basis of Transfer learning.
- So for our project we would be making use of the ResNet50 [ImageNet] Model.



Reference

ResNet :

- ResNet, short for Residual Networks is a classic neural network used as a backbone for many computer vision tasks
- The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with 150+layers successfully. Prior to ResNet training very deep neural networks was difficult due to the problem of vanishing gradients.
- ResNet uses skip connection to add the output from an earlier layer to a later layer. This helps it mitigate the vanishing gradient problem
- We make use of the ResNet50 present in Keras that is pretrained.



Skip Connection Image from DeepLearning.AI

Reference

K-Nearest Neighbours :

- The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.
- The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.
- “Birds of a feather flock together.

Advantages

- The algorithm is simple and easy to implement.
- There’s no need to build a model, tune several parameters, or make additional assumptions.
- The algorithm is versatile. It can be used for classification, regression, and search (as we will see in the next section)

Disadvantages

- The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.

The KNN Algorithm

1. Load the data
2. Initialize K to your chosen number of neighbors

3. For each example in the data

3.1 Calculate the distance between the query example and the current example from the data.

3.2 Add the distance and the index of the example to an ordered collection

4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances

5. Pick the first K entries from the sorted collection

6. Get the labels of the selected K entries

7. If regression, return the mean of the K labels

8. If classification, return the mode of the K labels

Cosine Similarity :

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis.

1. *Cosine similarity*: This measures the similarity using the cosine of the angle between two vectors in a multidimensional space. It is given by:

$$\text{similarity}\,(x,y) = \cos\,(\theta) = \frac{x \cdot y}{|x||y|} \tag{8.2}$$

2. *Euclidean distance*: This is the most common similarity distance measure and measures the distance between any two points in a euclidean space. It is given by:

$$d\,(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{8.3}$$

Euclidean Distance :

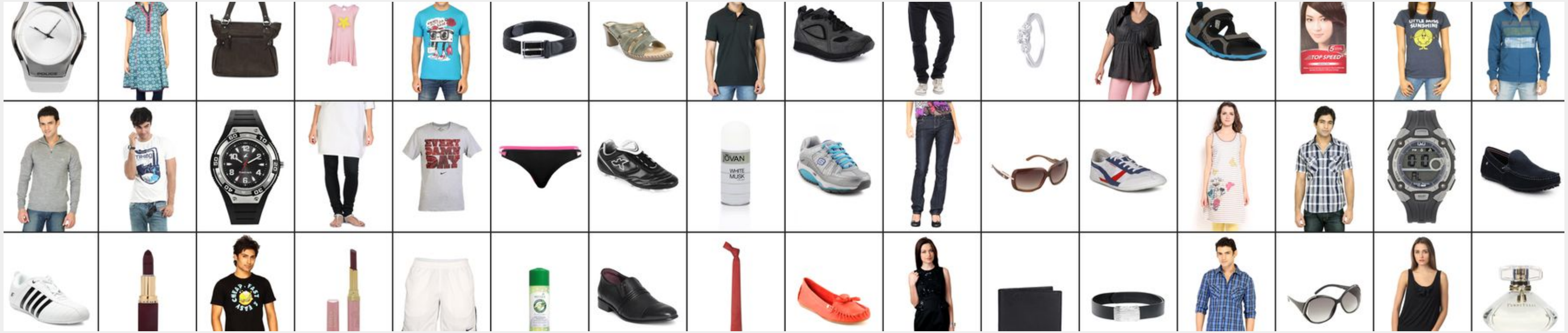
In mathematics, the Euclidean distance between two points in Euclidean space is the length of a line segment between the two points. It can be calculated from the Cartesian coordinates of the points using the Pythagorean theorem, therefore occasionally being called the Pythagorean distance.

About the Dataset :

Source of the dataset

Has more than 44,000 images of various product images , like shirts , tshirts , watches , sneakers , sarees.

The images are in high resolution but due to the limitations in computation power we would be making use of the low resolution images for faster computation and less storage.



02

PROJECT REQUIREMENTS

Tools and Technologies Used :

- Python 3.9
- Tensorflow
- Keras
- Scikit-Learn
- ResNet
- Streamlit
- Pycharm IDE
- Tqdm
- Numpy
- Pickle

03

CODE EXPLANATION

main.py

```python
import streamlit as st # to host system on web.
import os
from PIL import Image
import numpy as np
import pickle
import tensorflow
from tensorflow.keras.preprocessing import image
from tensorflow.keras.layers import GlobalMaxPooling2D
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
from sklearn.neighbors import NearestNeighbors
from numpy.linalg import norm

feature_list = np.array(pickle.load(open('embeddings.pkl', 'rb')))
filenames = pickle.load(open('filenames.pkl', 'rb'))

model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
model.trainable = False

model = tensorflow.keras.Sequential([
    model,
    GlobalMaxPooling2D()
])

st.title('Fashion Recommender System')
```

main.py

```python
# to upload the file - this code is required
def save_uploaded_file(uploaded_file):
    try:
        with open(os.path.join('uploads', uploaded_file.name), 'wb') as f:
            f.write(uploaded_file.getbuffer())
        return 1
    except:
        return 0

# calling the feature extraction function for the uploaded image.
def feature_extraction(img_path, model):
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    expanded_img_array = np.expand_dims(img_array, axis=0)
    preprocessed_img = preprocess_input(expanded_img_array)
    result = model.predict(preprocessed_img).flatten()
    normalized_result = result / norm(result)
    return normalized_result

# recommendation
def recommend(features, feature_list):
    neighbors = NearestNeighbors(n_neighbors=6, algorithm='brute', metric='euclidean')
    neighbors.fit(feature_list)
    distances, indices = neighbors.kneighbors([features])
    return indices
```

main.py

```python
# steps
# file upload -> save
uploaded_file = st.file_uploader("Choose an image")
print(uploaded_file.name)
if uploaded_file is not None:
    if save_uploaded_file(uploaded_file):
        # display the file
        display_image = Image.open(uploaded_file)
        st.image(display_image)
        # feature extract
        features = feature_extraction(os.path.join("uploads", uploaded_file.name), model)
        # st.text(features)
        # recommendention
        indices = recommend(features, feature_list)
        # show
        col1, col2, col3, col4, col5 = st.beta_columns(5)

        with col1:
            st.image(filenames[indices[0][0]])
        with col2:
            st.image(filenames[indices[0][1]])
        with col3:
            st.image(filenames[indices[0][2]])
        with col4:
            st.image(filenames[indices[0][3]])
        with col5:
            st.image(filenames[indices[0][4]])
    else:
        st.header("Some error occurred in file upload")

```

app.py

```python
import tensorflow
# image module to access functionalities related to images.
from tensorflow.keras.preprocessing import image
from tensorflow.keras.layers import GlobalMaxPooling2D
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
import numpy as np
from numpy.linalg import norm
import os
# tqdm - for knowing the progress of the for loop
from tqdm import tqdm
import pickle

model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
model.trainable = False

model = tensorflow.keras.Sequential([
    model,
    GlobalMaxPooling2D()
])

# print(model.summary())
```

app.py

```python
def extract_features(img_path, model):
    # loading the image
    img = image.load_img(img_path, target_size=(224, 224))

    # converting the image to an array - 224 * 224 * 3 - since RGB image
    img_array = image.img_to_array(img)

    # to reshape the images - input= image array , output=batch of images.
    expanded_img_array = np.expand_dims(img_array, axis=0)

    # preprocesses the input - by sending it to the function - preprocess_input - converts the input to the
    # appropriate input required by resnet
    # output given by it - Image is converted from RGB to BGR and other preprocessing is done for the resnet model.
    preprocessed_img = preprocess_input(expanded_img_array)

    # pass the image to resnet and convert it to one dimensional
    result = model.predict(preprocessed_img).flatten()

    # normalize the value from 0 to 1
    normalized_result = result / norm(result)

    return normalized_result
```

app.py

```
filenames = []

# for the images - we are appending the path name for the filenames.
for file in os.listdir('images'):
    filenames.append(os.path.join('images', file))

feature_list = []

for file in tqdm(filenames):
    feature_list.append(extract_features(file, model))

pickle.dump(feature_list, open('embeddings.pkl', 'wb'))
pickle.dump(filenames, open('filenames.pkl', 'wb'))
```

test.py

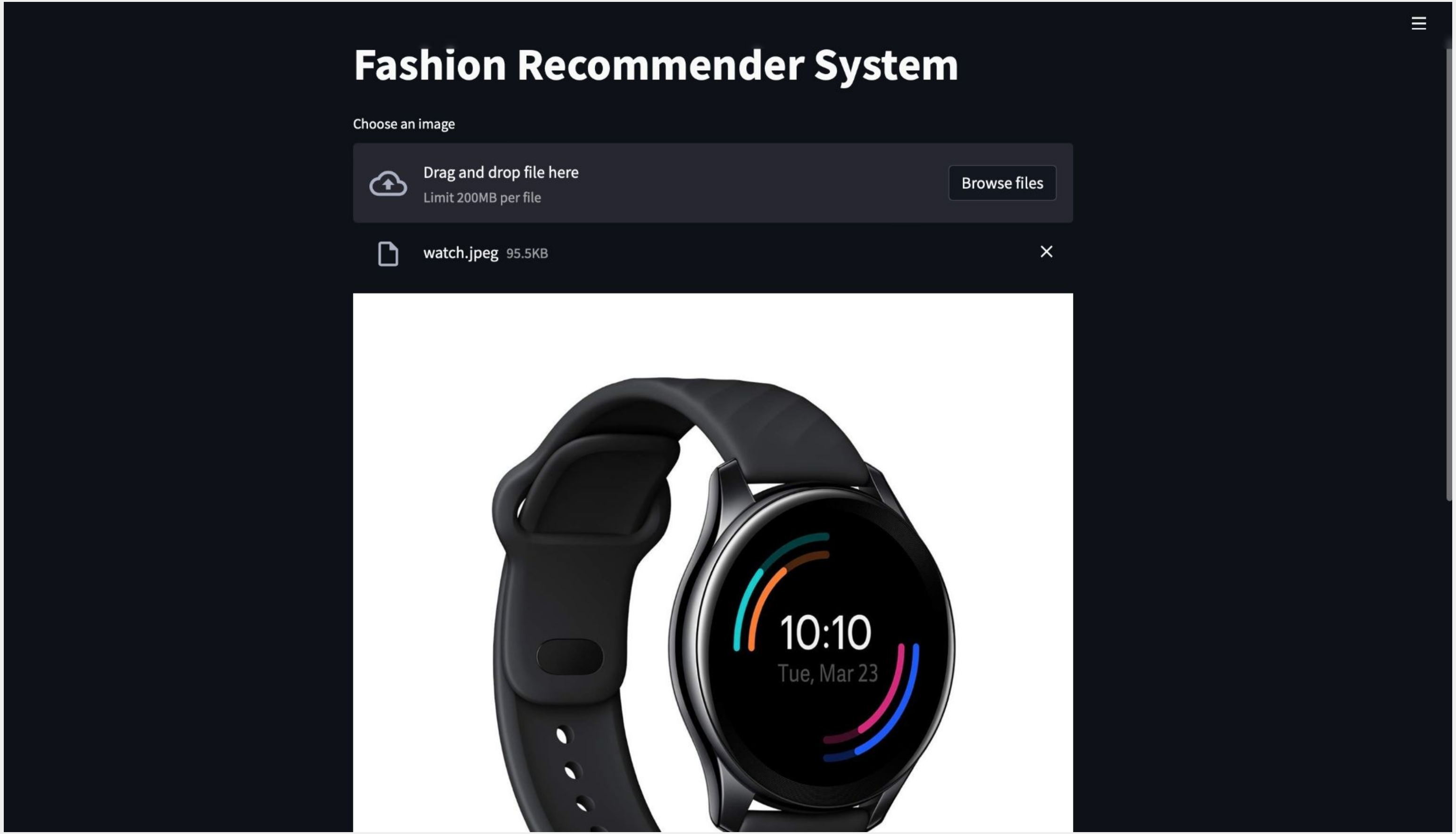
```python
import pickle
import tensorflow
import numpy as np
from numpy.linalg import norm
from tensorflow.keras.preprocessing import image
from tensorflow.keras.layers import GlobalMaxPooling2D
from tensorflow.keras.applications.resnet50 import ResNet50,preprocess_input
from sklearn.neighbors import NearestNeighbors
import cv2

# loading the feature list and filenames
feature_list = np.array(pickle.load(open('embeddings.pkl','rb')))
filenames = pickle.load(open('filenames.pkl','rb'))


model = ResNet50(weights='imagenet',include_top=False,input_shape=(224,224,3))
model.trainable = False

model = tensorflow.keras.Sequential([
    model,
    GlobalMaxPooling2D()
])

# taking a new image and extracting features and passing it to the model.
img = image.load_img('sample/jersey.jpeg',target_size=(224,224))
img_array = image.img_to_array(img)
expanded_img_array = np.expand_dims(img_array, axis=0)
preprocessed_img = preprocess_input(expanded_img_array)
result = model.predict(preprocessed_img).flatten()
normalized_result = result / norm(result)
```

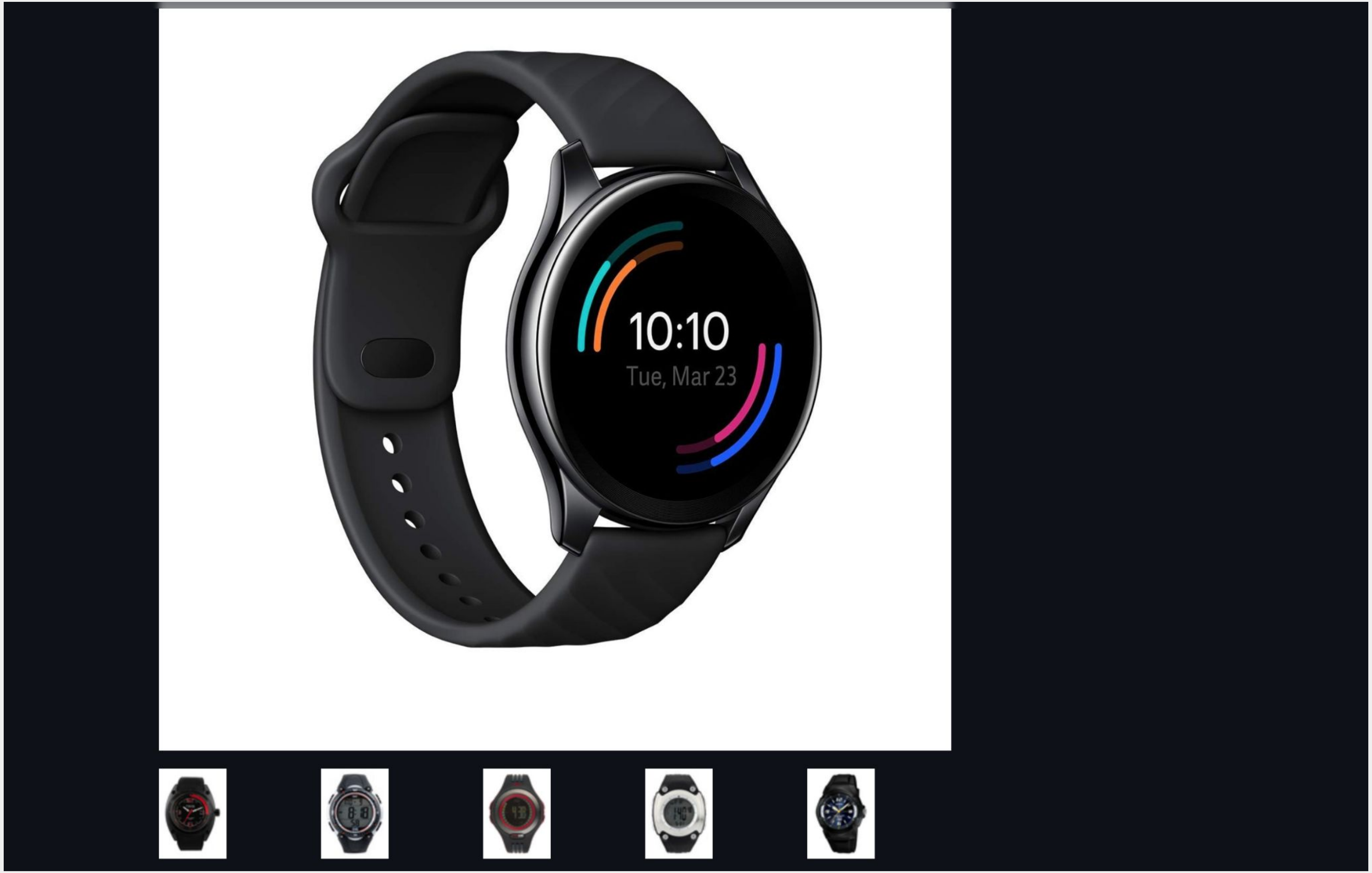
test.py

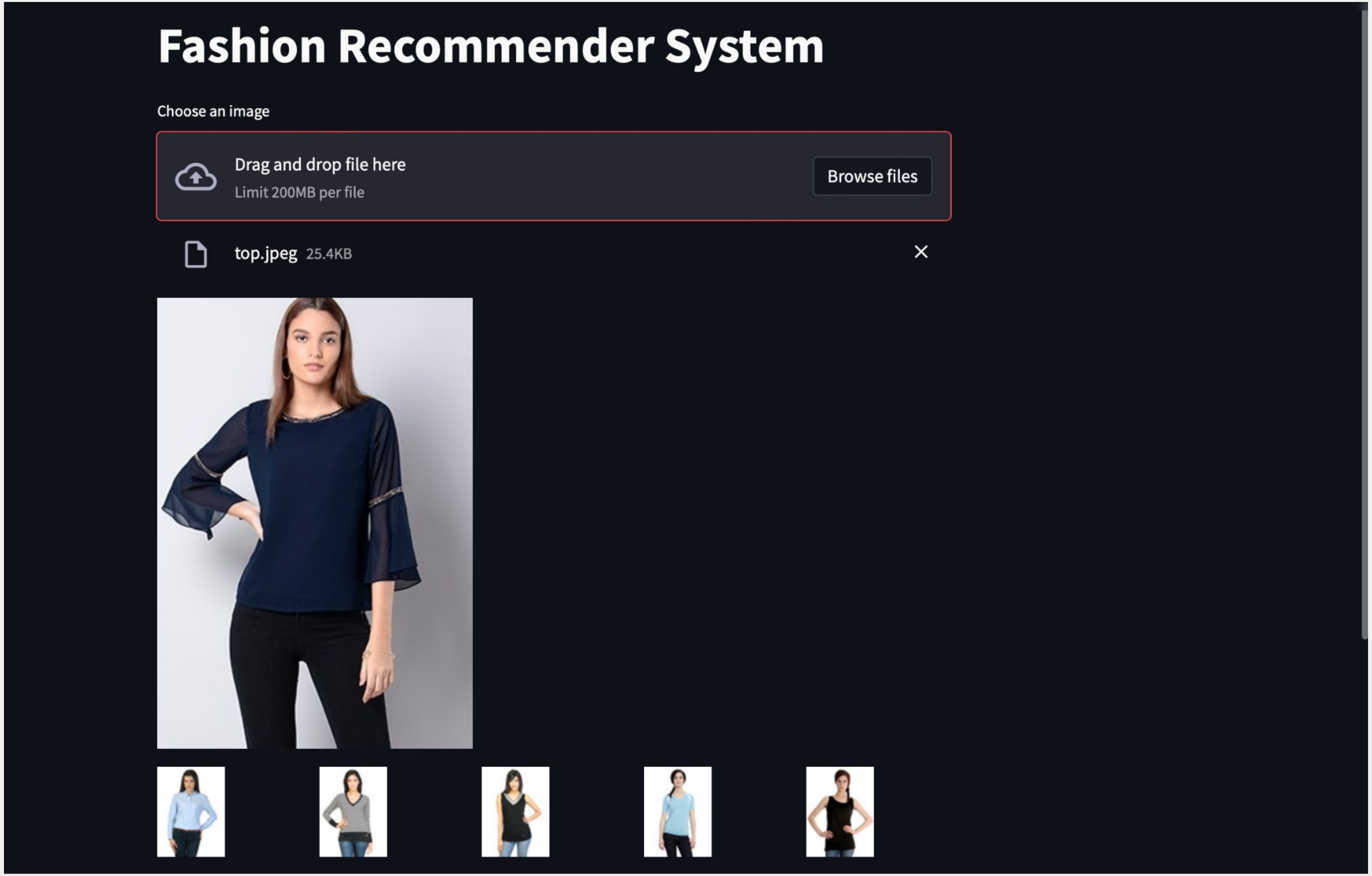
```python
# Finding the top-k nearest images using KNN
# Metric used = Euclidean Distance
# We can use cosine similarity as well.

neighbors = NearestNeighbors(n_neighbors=6,algorithm='brute',metric='euclidean')
#neighbors = NearestNeighbors(n_neighbors=6,algorithm='brute',metric='cosine')
# give the input data to .fit function
neighbors.fit(feature_list)

# finding the nearest k neighbours for the normalized result.
distances,indices = neighbors.kneighbors([normalized_result])

# printing the indices of the vectors which are nearest to the given image.
print(indices)

for file in indices[0][1:6]:
    temp_img = cv2.imread(filenames[file])
    # to display the image.
    cv2.imshow('output',cv2.resize(temp_img,(512,512)))
    # to make the screen wait.
    cv2.waitKey(0)

```

04

OUTPUTS


Fashion Recommender System
Choose an image
Drag and drop file here
Limit 200MB per file
Browse files
watch.jpeg 95.5KB
10:10
Tue, Mar 23


10:10
Tue, Mar 23


Fashion Recommender System
Choose an image
Drag and drop file here
Limit 200MB per file
Browse files
top.jpeg 25.4KB

Fashion Recommender System

Choose an image

Drag and drop file here
Limit 200MB per file

Browse files

sneakers.webp 58.8KB

THANK YOU

Github Link :
https://github.com/Priya2410/Fashion-Recommender-System-AIWIR-Project