DEPI Graduation project

Multimodal classification model

Multimodal classification model

This project is a hybrid machine learning system for clothing style classification, combining both Convolutional Neural Networks (**CNNs**) and **LSTM** to achieve highly accurate and context-aware predictions of outfit styles (e.g., Casual, Ethnic, Sport, Formal).

رواد مصر الرقمية

وزارة الاتصالات
وتكنولوجيا المعلومات

Team Members And Their Roles

Abanoub Ayman Nakhla

EDA and data cleaning

Ali Ahmed Adel El-Gamal

Model optimization and deployment

Mohamed Ahmed Adel

Model building and training

Sabry Wael Sabry

Visualization and model evaluation

Samir Nabil Samir

Documentation and deployment

Sohaib Mahmoud Mashaly

Data preprocessing and slideshow

Project Explanation

The project is a combination between 2 different ML model

1. One for analyzing the photos (**CNN**)
2. One for giving out the text classification for the attached photo **LSTM**

Project Explanation

Loading data

```python
# loading data
data = load_dataset('ashraq/fashion-product-images-small')
df = data['train'].to_pandas()
df
```

Python

	id	gender	masterCategory	subCategory	articleType	baseColour	season	year	usage	productDisplayName	image
0	15970	Men	Apparel	Topwear	Shirts	Navy Blue	Fall	2011.0	Casual	Turtle Check Men Navy Blue Shirt	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
1	39386	Men	Apparel	Bottomwear	Jeans	Blue	Summer	2012.0	Casual	Peter England Men Party Blue Jeans	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
2	59263	Women	Accessories	Watches	Watches	Silver	Winter	2016.0	Casual	Titan Women Silver Watch	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
3	21379	Men	Apparel	Bottomwear	Track Pants	Black	Fall	2011.0	Casual	Manchester United Men Solid Black Track Pants	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
4	53759	Men	Apparel	Topwear	Tshirts	Grey	Summer	2012.0	Casual	Puma Men Grey T-shirt	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
...	...	...	...	...	...	...	...	...	...	...	...

Project Explanation

Text classification model

```python
# dropping id column
df.drop('id', axis=1, inplace=True)
df.head()
```

Python

	gender	masterCategory	subCategory	articleType	baseColour	season	year	usage	productDisplayName	image
0	Men	Apparel	Topwear	Shirts	Navy Blue	Fall	2011.0	Casual	Turtle Check Men Navy Blue Shirt	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
1	Men	Apparel	Bottomwear	Jeans	Blue	Summer	2012.0	Casual	Peter England Men Party Blue Jeans	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
2	Women	Accessories	Watches	Watches	Silver	Winter	2016.0	Casual	Titan Women Silver Watch	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
3	Men	Apparel	Bottomwear	Track Pants	Black	Fall	2011.0	Casual	Manchester United Men Solid Black Track Pants	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
4	Men	Apparel	Topwear	Tshirts	Grey	Summer	2012.0	Casual	Puma Men Grey T-shirt	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...

Project Explanation

EDA

```python
# printing some basic information about the data
df.info()
```

Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44072 entries, 0 to 44071
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   gender              44072 non-null  object
 1   masterCategory      44072 non-null  object
 2   subCategory         44072 non-null  object
 3   articleType         44072 non-null  object
 4   baseColour          44072 non-null  object
 5   season              44072 non-null  object
 6   year                44072 non-null  float64
 7   usage               44072 non-null  object
 8   productDisplayName  44072 non-null  object
 9   image               44072 non-null  object
dtypes: float64(1), object(9)
memory usage: 3.4+ MB
```

Project Explanation

EDA

```
# checking for missing values
df.isnull().sum()
```

	0
gender	0
masterCategory	0
subCategory	0
articleType	0
baseColour	0
season	0
year	0
usage	0
productDisplayName	0
image	0

dtype: int64

Project Explanation

EDA

```
# checking for duplicates with skipping image column
df.drop('image', axis=1).duplicated().sum()
```

```
np.int64(10754)
```

Project Explanation

EDA

```
    # checking categorical columns values
    for col in df.drop(columns = ['productDisplayName' , 'image']).select_dtypes(include='object').columns:
        print(f'{col} : {df[col].unique()}')
        print('-----------------------------------------------------------')
```

```
gender : ['Men' 'Women' 'Boys' 'Girls' 'Unisex']
-----------------------------------------------------------
masterCategory : ['Apparel' 'Accessories' 'Footwear' 'Personal Care' 'Free Items'
 'Sporting Goods' 'Home']
-----------------------------------------------------------
subCategory : ['Topwear' 'Bottomwear' 'Watches' 'Socks' 'Shoes' 'Belts' 'Flip Flops'
 'Bags' 'Innerwear' 'Sandal' 'Shoe Accessories' 'Fragrance' 'Jewellery'
 'Lips' 'Saree' 'Eyewear' 'Scarves' 'Dress' 'Loungewear and Nightwear'
 'Wallets' 'Apparel Set' 'Headwear' 'Mufflers' 'Skin Care' 'Makeup'
 'Free Gifts' 'Ties' 'Accessories' 'Nails' 'Beauty Accessories'
 'Water Bottle' 'Skin' 'Eyes' 'Bath and Body' 'Gloves'
 'Sports Accessories' 'Cufflinks' 'Sports Equipment' 'Stoles' 'Hair'
 'Perfumes' 'Home Furnishing' 'Umbrellas' 'Wristbands' 'Vouchers']
```

Project Explanation

EDA

```
# checking number of unique values in each column
for col in df.drop(columns = [ 'image']).columns:
    print(f'{col} : {df[col].nunique()}')
    print('---------------------------------------------------------------')
```

```
gender : 5
---------------------------------------------------------------
masterCategory : 7
---------------------------------------------------------------
subCategory : 45
---------------------------------------------------------------
articleType : 141
---------------------------------------------------------------
baseColour : 46
---------------------------------------------------------------
season : 4
---------------------------------------------------------------
year : 13
---------------------------------------------------------------
usage : 8
---------------------------------------------------------------
productDisplayName : 30801
---------------------------------------------------------------
```

Project Explanation

EDA

```
# checking images duplicates
import hashlib

# converting images into hash values to recognise duplicates
def hash_image(img_bytes):
    return hashlib.md5(img_bytes).hexdigest()

# Extract raw bytes and check for duplicates
df["image_hash"] = df["image"].apply(lambda x: hash_image(x["bytes"]))
df['image_hash'].duplicated().sum()
```

```
np.int64(778)
```

- number of duplicates in image is lower than the text in the data

Project Explanation

EDA

```
# checking duplicates for both text data and images togehter
df.drop('image', axis=1).duplicated().sum()
```

```
np.int64(540)
```

- so the final thoughts is that there is only 540 duplicated rows

Project Explanation

EDA

```
# checking class balance
df.usage.value_counts()
```

usage	count
Casual	34392
Sports	4004
Ethnic	3208
Formal	2345
Smart Casual	67
Party	29
Travel	26
Home	1

Project Explanation

Data Cleaning

```
# drop duplicated rows
duplicated_rows = df[df.drop(columns='image').duplicated()].index
df.drop( index=duplicated_rows , inplace=True)
df
```

	gender	masterCategory	subCategory	articleType	baseColour	season	year	usage	productDisplayName
0	Men	Apparel	Topwear	Shirts	Navy Blue	Fall	2011.0	Casual	Turtle Check Men Navy Blue Shirt
1	Men	Apparel	Bottomwear	Jeans	Blue	Summer	2012.0	Casual	Peter England Men Party Blue Jeans
2	Women	Accessories	Watches	Watches	Silver	Winter	2016.0	Casual	Titan Women Silver Watch
3	Men	Apparel	Bottomwear	Track Pants	Black	Fall	2011.0	Casual	Manchester United Men Solid Black Track Pants
4	Men	Apparel	Topwear	Tshirts	Grey	Summer	2012.0	Casual	Puma Men Grey T-shirt
...	...	...	...	...	...	...	...	...	...
44067	Men	Footwear	Shoes	Casual Shoes	White	Summer	2013.0	Casual	Gas Men Caddy Casual Shoe
44068	Men	Footwear	Flip Flops	Flip Flops	Red	Summer	2011.0	Casual	Lotto Men's Soccer Track Flip Flop

Project Explanation

Data Cleaning

```
# dropping low samples classes depending on usage column
small_classes = ['Smart Casual' , 'Party' , 'Travel' , 'Home']
df = df[~df['usage'].isin(small_classes)]
df
```

	gender	masterCategory	subCategory	articleType	baseColour	season	year	usage	productDisplayName
0	Men	Apparel	Topwear	Shirts	Navy Blue	Fall	2011.0	Casual	Turtle Check Men Navy Blue Shirt
1	Men	Apparel	Bottomwear	Jeans	Blue	Summer	2012.0	Casual	Peter England Men Party Blue Jeans
2	Women	Accessories	Watches	Watches	Silver	Winter	2016.0	Casual	Titan Women Silver Watch
3	Men	Apparel	Bottomwear	Track Pants	Black	Fall	2011.0	Casual	Manchester United Men Solid Black Track Pants
4	Men	Apparel	Topwear	Tshirts	Grey	Summer	2012.0	Casual	Puma Men Grey T-shirt
...	...	...	...	...	...	...	...	...	...
44067	Men	Footwear	Shoes	Casual Shoes	White	Summer	2013.0	Casual	Gas Men Caddy Casual Shoe

Project Explanation

Data Cleaning

```
# balancing the classes using under sampling
from sklearn.utils import resample
# Separate majority and minority classes
majority_class = df[df.usage == 'Casual']
minority_classes = df[df.usage != 'Casual']
# Downsample majority class
majority_downsampled = resample(majority_class,
                                replace=False,    # sample without replacement
                                n_samples=minority_classes.usage.value_counts().max(), # to match minority class
                                random_state=42) # reproducible results
# Combine minority classes with downsampled majority class
df_balanced = pd.concat([majority_downsampled, minority_classes])
df_balanced.usage.value_counts()
```

usage	count
Casual	4004
Sports	4004
Ethnic	3206
Formal	2285

dtype: int64

Project Explanation

Data Cleaning

```
df_balanced.reset_index(drop=True, inplace=True)
df_balanced
```

	gender	masterCategory	subCategory	articleType	baseColour	season	year	usage	productDisplayName	image
0	Men	Footwear	Shoes	Casual Shoes	Black	Fall	2011.0	Casual	iD Men Casual Black Shoes	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
1	Men	Apparel	Topwear	Shirts	White	Summer	2012.0	Casual	Denizen Men White Shirt	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
2	Unisex	Accessories	Watches	Watches	Yellow	Winter	2016.0	Casual	Q&Q Kids Unisex White Dial Analog Watch	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
3	Men	Apparel	Topwear	Tshirts	Pink	Fall	2011.0	Casual	Flying Machine Men Damson Maroon Tshirts	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
4	Men	Apparel	Innerwear	Innerwear Vests	Black	Summer	2016.0	Casual	Levis Men Black & White Innerwear Vest	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
...	...	...	...	...	...	...	...	...	...	...
13494	Men	Apparel	Bottomwear	Shorts	White	Spring	2013.0	Sports	Nike Men As Woven Shor White Shorts	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
13495	Men	Apparel	Topwear	Tshirts	Black	Summer	2012.0	Sports	Nike Men Black T-shirt	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
13496	Men	Footwear	Shoes	Sports Shoes	White	Summer	2011.0	Sports	Fila Men's Leonard White Black Shoe	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
13497	Men	Apparel	Topwear	Tshirts	Red	Fall	2010.0	Sports	Kipsta Men Loose Fit Round Neck Jersey Red	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...
13498	Men	Apparel	Topwear	Tshirts	Black	Summer	2012.0	Sports	Nike Men Striped Black Jersey	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...

Project Explanation

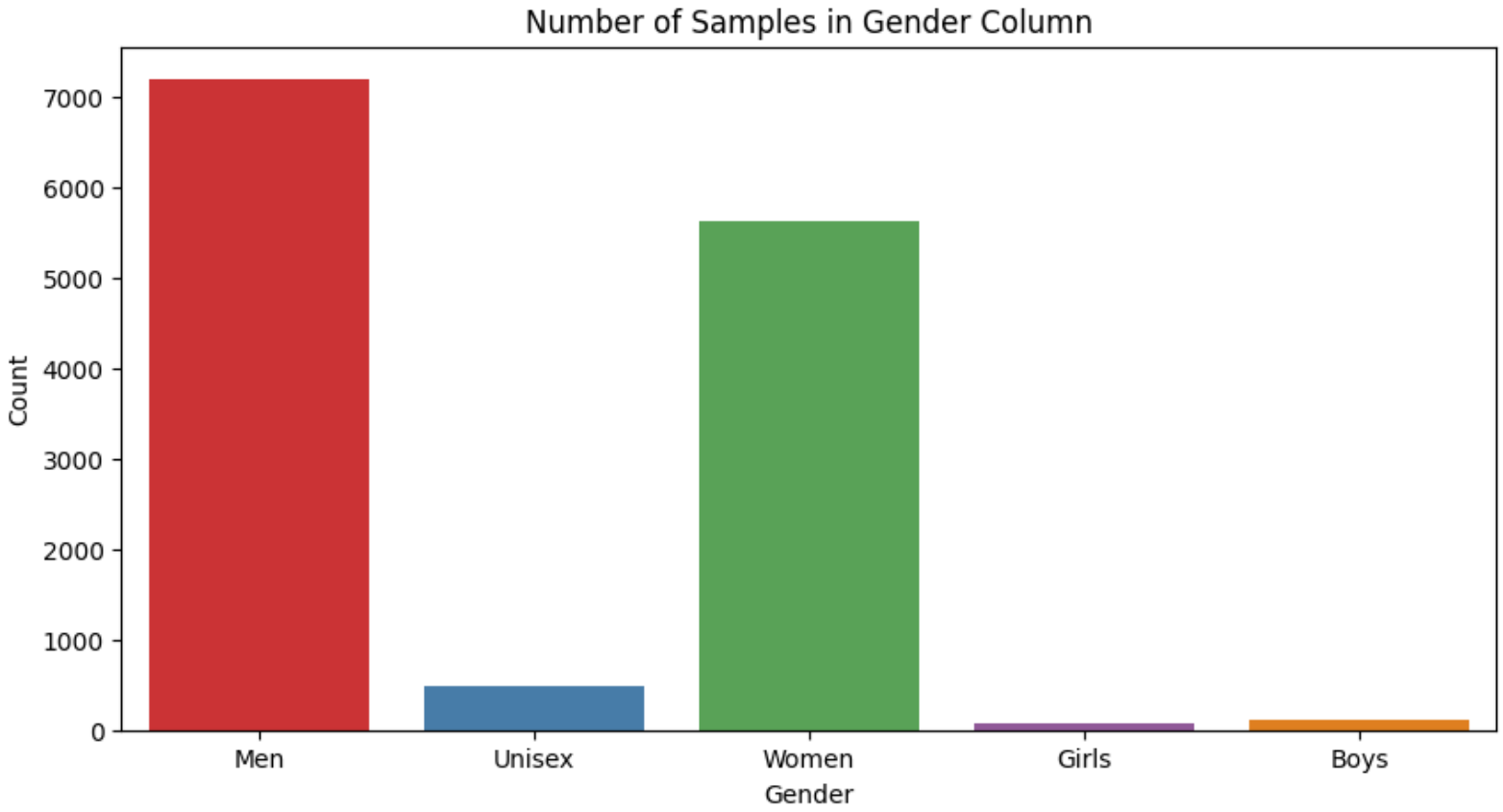
Data Cleaning

```python
# getting image pixels values
import io
from PIL import Image

def get_img_pixels(img_dict):
    img_bytes = img_dict["bytes"]
    # Convert bytes → PIL Image
    img = Image.open(io.BytesIO(img_bytes))

    # Convert PIL Image → numpy array of pixels
    pixels = np.array(img)
    return pixels

df_balanced['image_pixels'] = df_balanced['image'].apply(get_img_pixels)
df_balanced['image_pixels']
```

	image_pixels
0	[[[255, 255, 255], [255, 255, 255], [255, 255,...
1	[[[255, 255, 255], [255, 255, 255], [255, 255,...
2	[[[255, 255, 255], [255, 255, 255], [255, 255,...
3	[[[255, 255, 255], [255, 255, 255], [255, 255,...
4	[[[255, 255, 255], [255, 255, 255], [255, 255,...

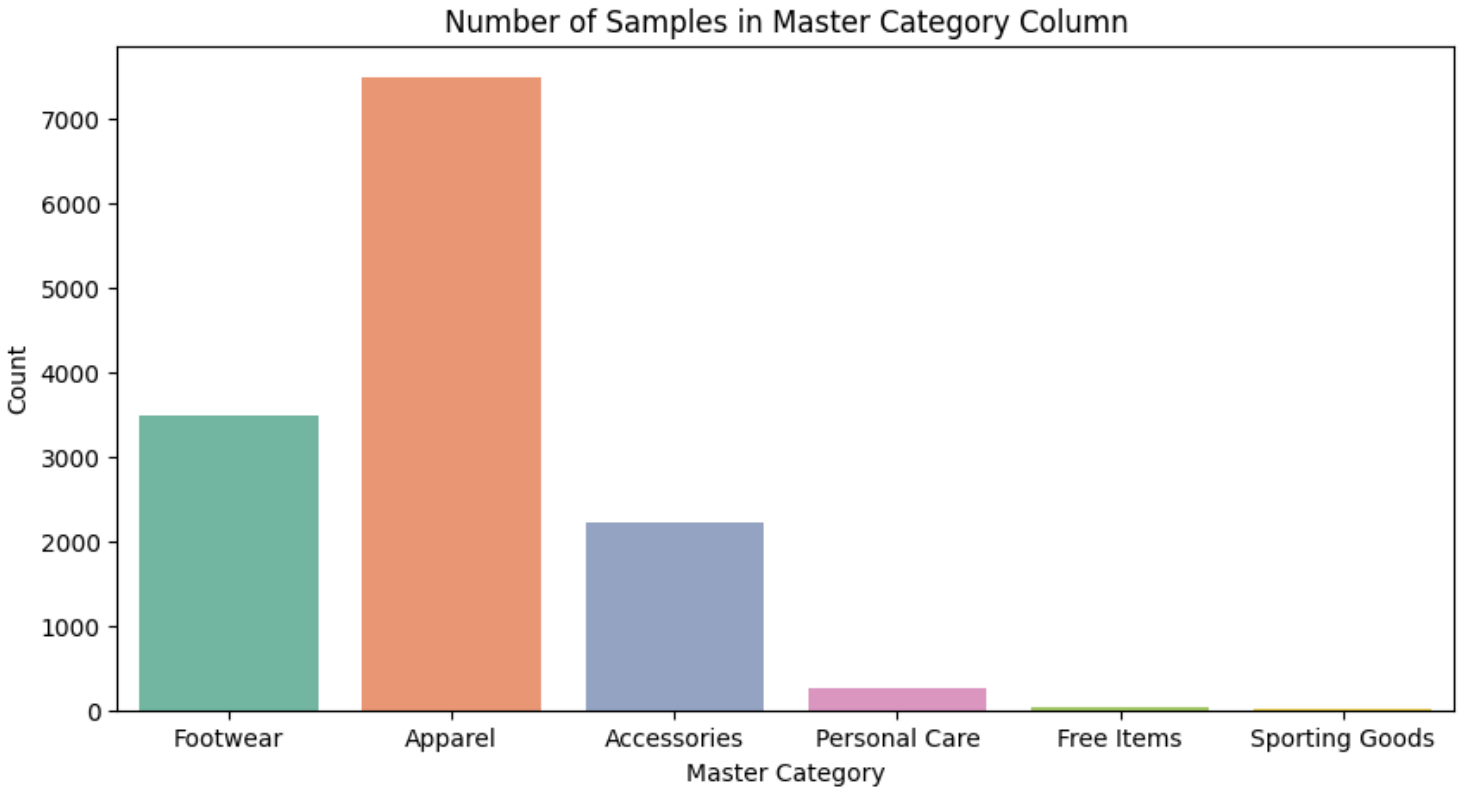
Project Explanation

Visualizations



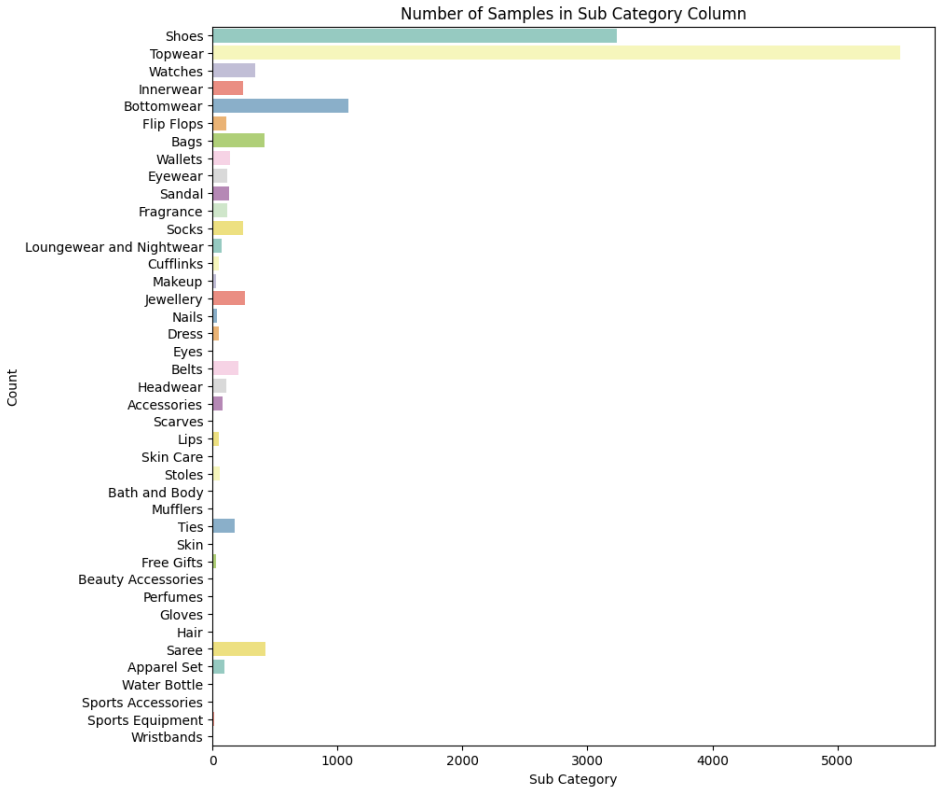
Project Explanation

Visualizations



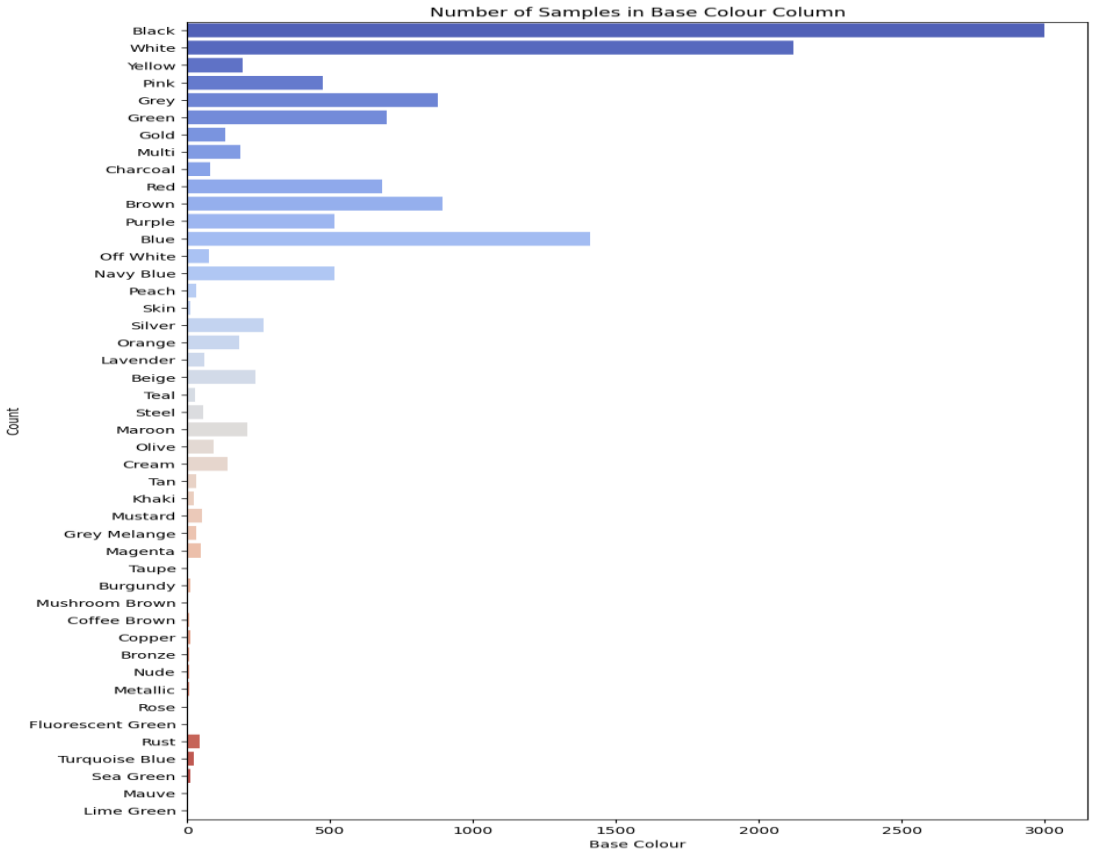
Project Explanation

Visualizations



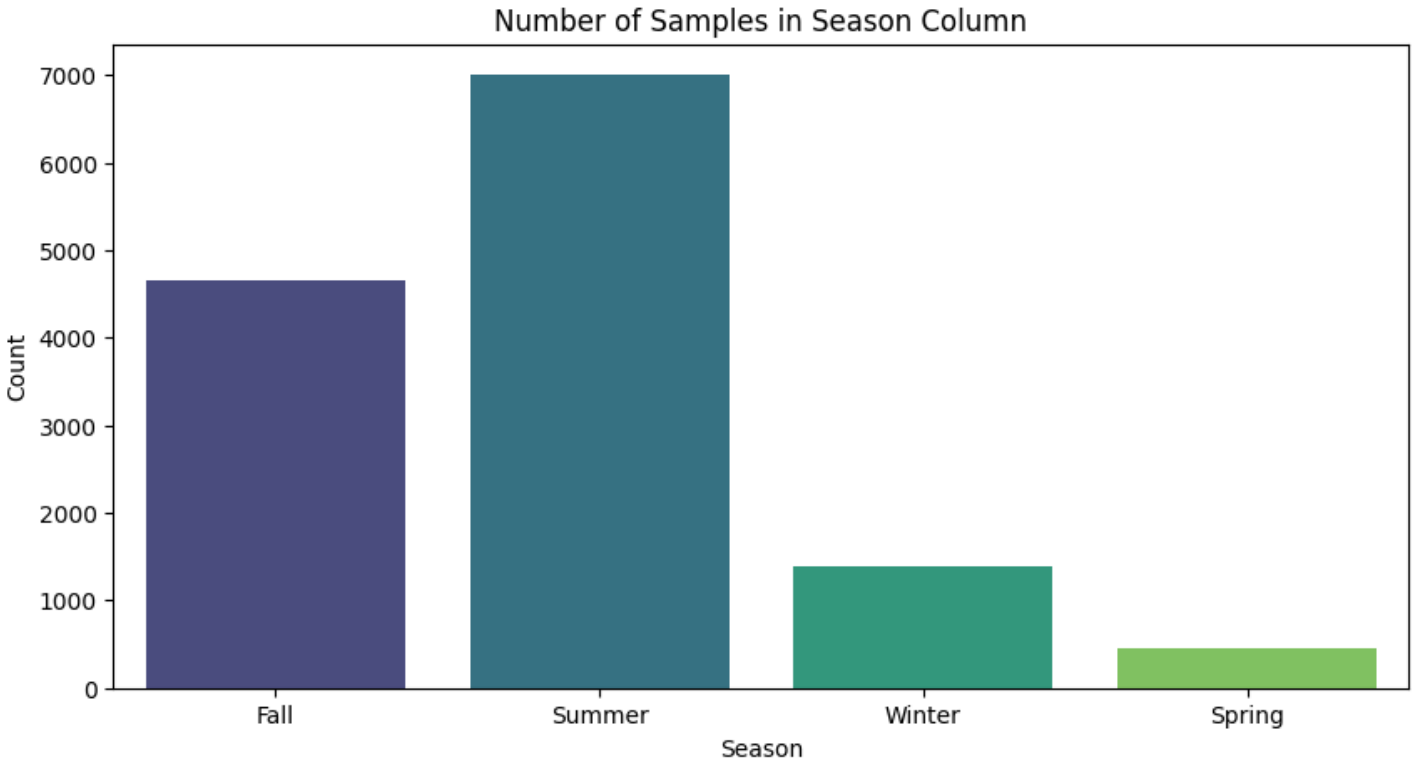
Project Explanation

Visualizations



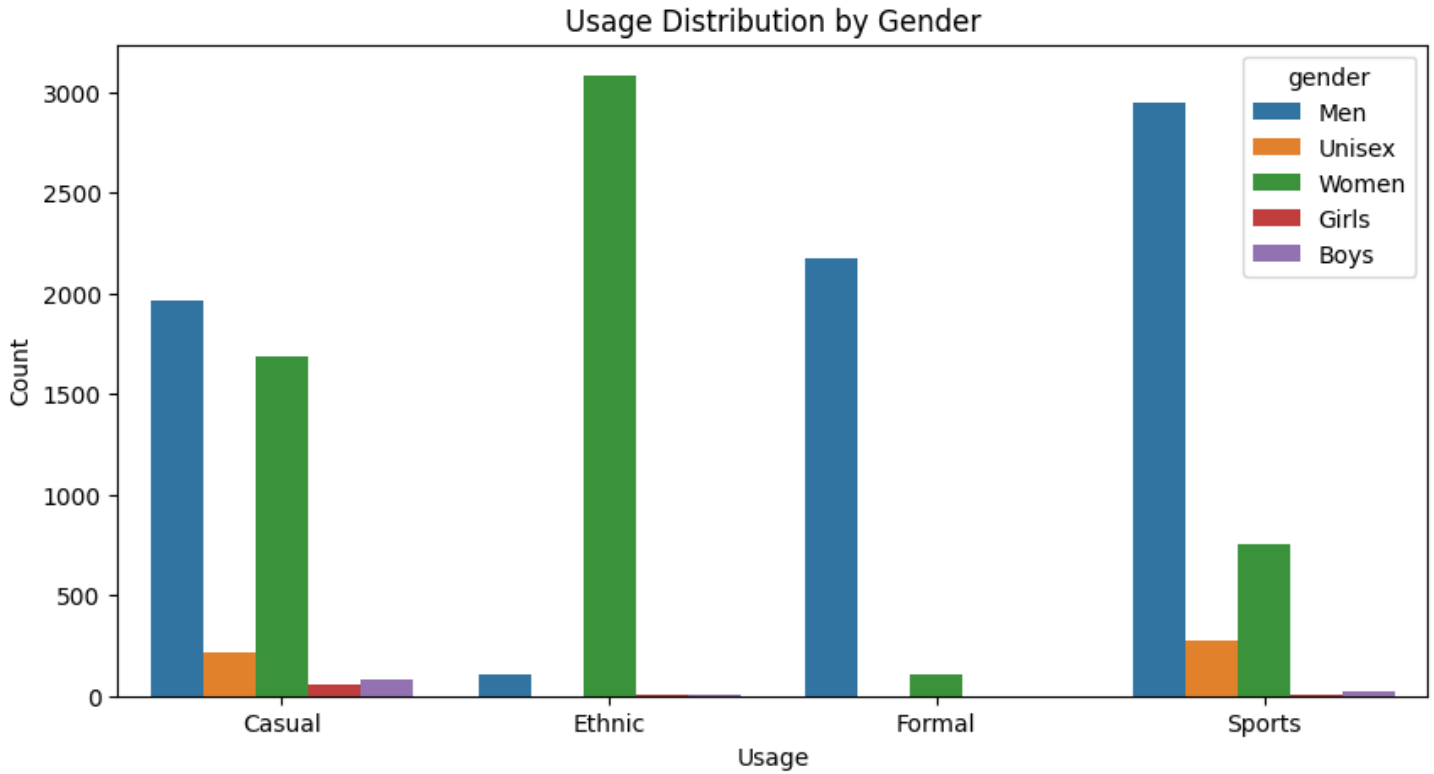
Project Explanation

Visualizations



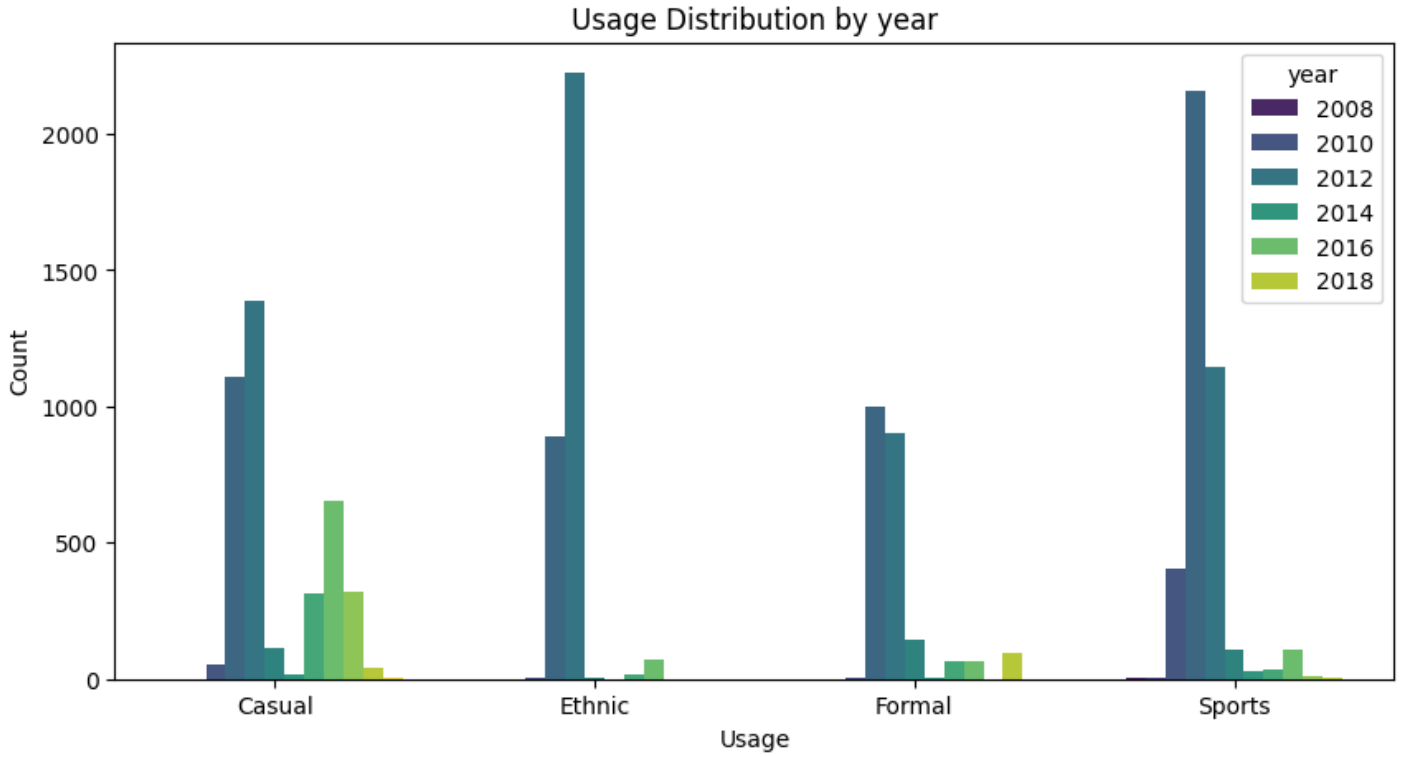
Project Explanation

Visualizations



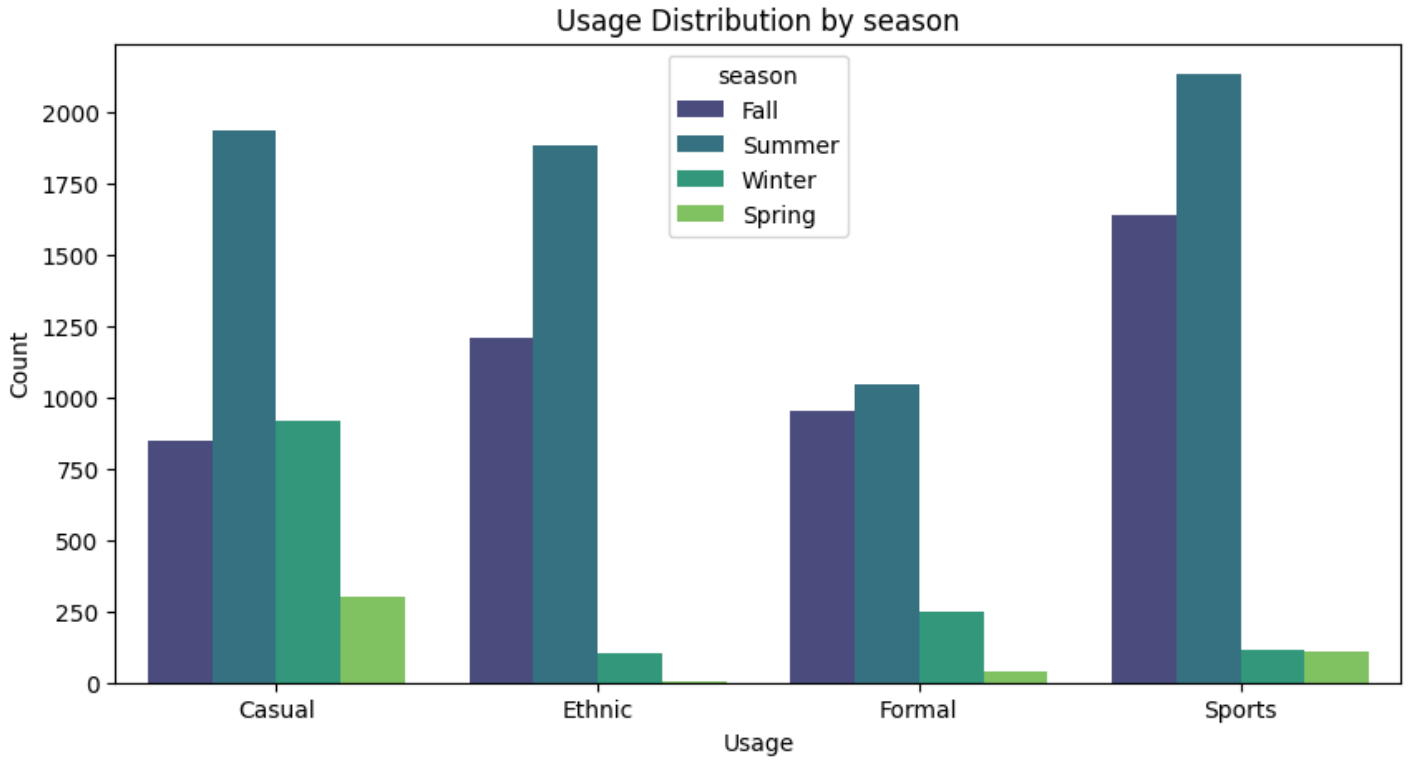
Project Explanation

Visualizations



Project Explanation

Visualizations



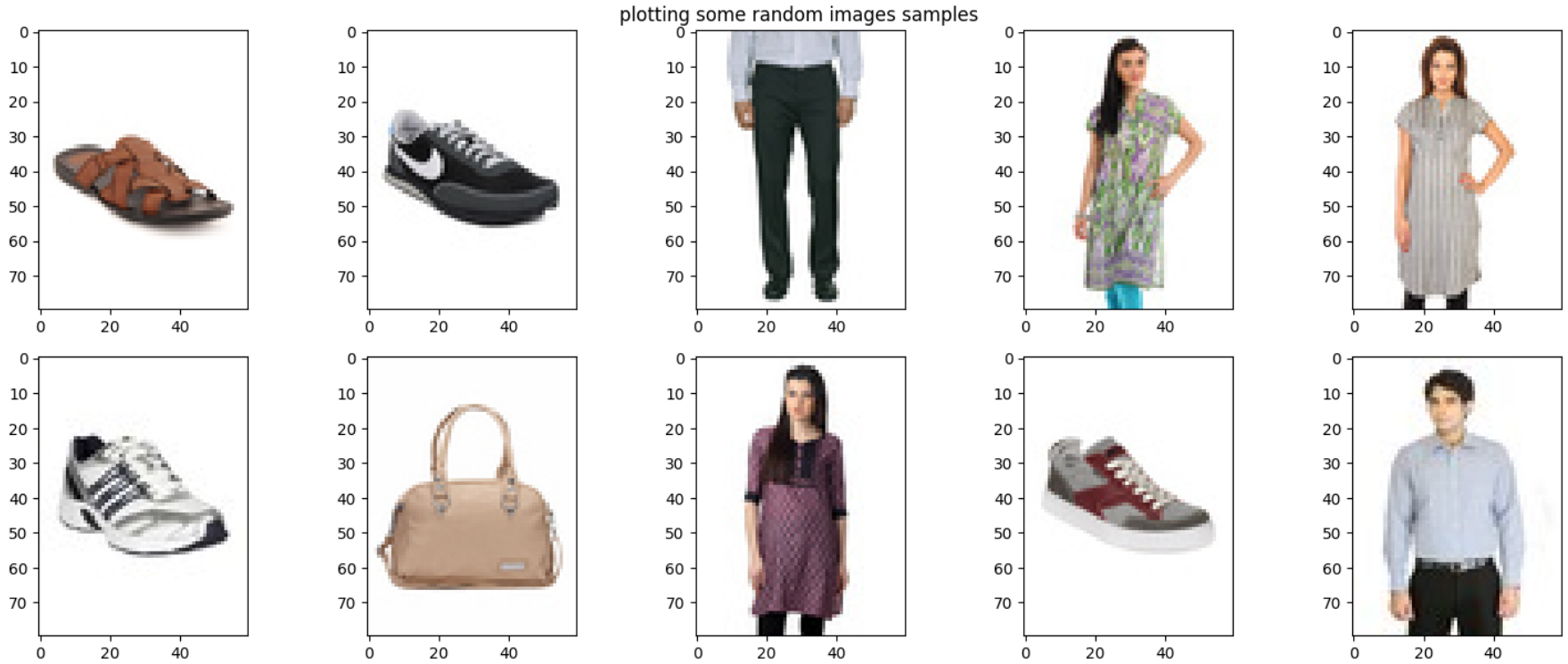
Project Explanation

Visualizations

Word Clous for the most common words

Project Explanation

Visualizations



Project Explanation

Splitting The Data

splitting data

- we'll be working with only 3 columns:
- `image pixels`
- `product display name`
- `usage`

```
# splitting target from features
x = df_balanced[['productDisplayName' , 'image_pixels']]
y = df_balanced['usage']
```

```
# splitting data into train and test
from sklearn.model_selection import train_test_split

x_train , x_test , y_train , y_test = train_test_split(x , y , test_size=0.2 , random_state=42 , stratify=y)
```

Project Explanation

Data Preprocessing (Images)

data preprocessing

for images :

- make all photos of size 128x128
- improve image quality
- normalize all images to range from 0 to 1

```python
import cv2
def preprocess_image_debug(img):
    # print("\n===== TRAINING IMAGE DEBUG =====")
    # print("RAW image type:", type(img))

    # ---- COPY THE REAL PREPROCESSING ----
    if img is None or not isinstance(img, np.ndarray):
        return None
```

Project Explanation

Data Preprocessing (Images)

```python
    # Ensure 3 channels
    if img.ndim == 2:
        img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)

    if img.shape[2] == 4:  # RGBA → RGB
        img = cv2.cvtColor(img, cv2.COLOR_BGRA2BGR)

    # Sharpen
    kernel = np.array([[0, -1, 0],
                       [-1, 5, -1],
                       [0, -1, 0]])
    img = cv2.filter2D(img, -1, kernel)

    # Resize
    img = cv2.resize(img, (128, 128), interpolation=cv2.INTER_AREA)

    # Normalize
    img = img.astype("float32") / 255.0

    print("AFTER preprocess shape:", img.shape)
    print("AFTER preprocess first pixel:", img[0, 0])

    return img
```

0.1s

Project Explanation

Data Preprocessing (Images)

```
# enhancing images
from joblib import Parallel, delayed
x_train['image_pixels'] = Parallel(n_jobs=4)(delayed(preprocess_image)(img) for img in x_train['image_pixels'])
x_test['image_pixels'] = Parallel(n_jobs=4)(delayed(preprocess_image)(img) for img in x_test['image_pixels'])
```

```
x_train.shape
```

```
(10799, 2)
```

Project Explanation

Data Preprocessing (Images)

for target :

- encoding using label encoder

```
# applying label encoder
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)
```

```
import joblib
joblib.dump(label_encoder, "label_encoder.joblib")
```

```
['label_encoder.joblib']
```

Project Explanation

Data Preprocessing (Text)

text preprocessing

- using pretrained model

```
# encoding the text using pretrained model (Bert)
from transformers import AutoTokenizer, AutoModel
import torch

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = AutoModel.from_pretrained("bert-base-uncased")
```

Project Explanation

Data Preprocessing (Text)

```python
from tqdm import tqdm

# Put model in evaluation mode
model.eval()

# Optional: Use GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Function to get mean pooled embedding
def get_embedding(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True).to(device)
    with torch.no_grad():
        outputs = model(**inputs)
    embedding = outputs.last_hidden_state.mean(dim=1).squeeze().cpu().numpy()
    return embedding
```

Project Explanation

Data Preprocessing (Text)

```
# Apply to DataFrame column
tqdm.pandas()  # for progress bar
x_train["productEmbedding"] = x_train["productDisplayName"].progress_apply(get_embedding)
x_test["productEmbedding"] = x_test["productDisplayName"].progress_apply(get_embedding)
```

```
100%|██████████| 10799/10799 [18:09<00:00,  9.91it/s]
100%|██████████| 2700/2700 [04:21<00:00, 10.32it/s]
```

Project Explanation

Data Preprocessing (Text)

```
x_train['productEmbedding']
```

	productEmbedding
9372	[0.72427547, -0.24629416, -0.52753526, -0.0616...
11118	[0.1917454, 0.017584328, -0.7149372, -0.147989...
13177	[0.20069282, -0.68254113, -0.1824408, 0.055097...
5025	[0.2761686, -0.122196525, -0.43160215, -0.3446...
1210	[0.49720854, -0.15911539, 0.4221167, 0.1660808...
...	...
5949	[0.501969, -0.07212414, -0.053167872, 0.049404...
8405	[0.03037176, 0.15981106, -0.49047953, -0.20095...
10654	[0.6375512, 0.11681496, -0.42582986, 0.0737109...
8663	[0.40547118, -0.47648674, -0.37352374, -0.2703...
9406	[-0.49867922, -0.13441913, 0.37491456, -0.2443...

10799 rows × 1 columns

Project Explanation

Building The Model

building model

first model

```
# libraries
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Input, Dense, Dropout,LSTM, Conv2D, MaxPooling2D, Flatten, GlobalAveragePooling2D, Concatenate
)
from tensorflow.keras import regularizers
```

Project Explanation

Building The Model

```
# TEXT BRANCH
text_input_1 = Input(shape=(1, 768), name='text_input')
x_text_1 = LSTM(64,kernel_regularizer=regularizers.l2(0.001))(text_input_1)
x_text_1 = Dropout(0.2)(x_text_1)
```

Project Explanation

Building The Model

```
# IMAGE BRANCH
image_input_1 = Input(shape=(128, 128, 3), name='image_input')
x_image_1 = Conv2D(32, (3, 3),kernel_regularizer=regularizers.l2(0.1), activation='relu')(image_input_1)
x_image_1 = MaxPooling2D((2, 2))(x_image_1)
x_image_1 = Conv2D(64, (3, 3), activation='relu')(x_image_1)
x_image_1 = MaxPooling2D((2, 2))(x_image_1)
x_image_1 = Flatten()(x_image_1)
x_image_1 = Dropout(0.5)(x_image_1)
```

Project Explanation

Building The Model

```
# CONCATENATE
combined_1 = Concatenate()([x_text_1, x_image_1])
x_1 = Dense(64, activation='relu')(combined_1)
output_1 = Dense(4, activation='softmax')(x_1)  # Use softmax for multi-class
```

Project Explanation

Building The Model

```
# MODEL
model_1 = Model(inputs=[text_input_1, image_input_1], outputs=output_1)
model_1.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model_1.summary()
```

Project Explanation

Building The Model

Layer (type)	Output Shape	Param #	Connected to
image_input (InputLayer)	(None, 128, 128, 3)	0	-
conv2d (Conv2D)	(None, 126, 126, 32)	896	image_input[0][0]
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18,496	max_pooling2d[0]…
text_input (InputLayer)	(None, 1, 768)	0	-
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0	conv2d_1[0][0]
lstm (LSTM)	(None, 64)	213,248	text_input[0][0]
flatten (Flatten)	(None, 57600)	0	max_pooling2d_1[…
dropout (Dropout)	(None, 64)	0	lstm[0][0]
dropout_1 (Dropout)	(None, 57600)	0	flatten[0][0]
concatenate (Concatenate)	(None, 57664)	0	dropout[0][0], dropout_1[0][0]
dense (Dense)	(None, 64)	3,690,560	concatenate[0][0]
dense_1 (Dense)	(None, 4)	260	dense[0][0]

Project Explanation

Building The Model

```
# computing class weights for better generalization
from sklearn.utils.class_weight import compute_class_weight

class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(y_train), y=y_train)
class_weight_dict = dict(enumerate(class_weights))
class_weight_dict
```

```
{0: np.float64(0.8428816734311583),
 1: np.float64(1.0525341130604289),
 2: np.float64(1.476887308533917),
 3: np.float64(0.8428816734311583)}
```

```
label_encoder.classes_
```

```
array(['Casual', 'Ethnic', 'Formal', 'Sports'], dtype=object)
```

Project Explanation

Training The Model

```
# training the model
# Step 1: Convert text to sequence embeddings (shape: [1, 768])
x_train_text_embeddings_1 = np.stack(x_train['productEmbedding'].values)
x_train_text_embeddings_1 = x_train_text_embeddings_1.reshape((-1, 1, 768))

# Step 2: Prepare image data
x_train_image_data_1 = np.stack(x_train['image_pixels'])

# Step 3: Train the model
history = model_1.fit(
    {'text_input': x_train_text_embeddings_1, 'image_input': x_train_image_data_1},
    y_train,
    epochs=10,
    batch_size=64,
    validation_split=0.2,
    class_weight=class_weight_dict
)
```

Project Explanation

Training The Model

```
Epoch 1/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 49s 340ms/step - accuracy: 0.7414 - loss: 1.0790 - val_accuracy: 0.8968 - val_loss: 0.5680
Epoch 2/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 43s 322ms/step - accuracy: 0.9006 - loss: 0.4818 - val_accuracy: 0.9037 - val_loss: 0.4353
Epoch 3/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 41s 305ms/step - accuracy: 0.9184 - loss: 0.3510 - val_accuracy: 0.9185 - val_loss: 0.3453
Epoch 4/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 39s 290ms/step - accuracy: 0.9287 - loss: 0.2817 - val_accuracy: 0.9144 - val_loss: 0.3144
Epoch 5/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 36s 264ms/step - accuracy: 0.9382 - loss: 0.2354 - val_accuracy: 0.9199 - val_loss: 0.2998
Epoch 6/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 35s 259ms/step - accuracy: 0.9483 - loss: 0.1922 - val_accuracy: 0.9181 - val_loss: 0.3043
Epoch 7/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 35s 258ms/step - accuracy: 0.9532 - loss: 0.1736 - val_accuracy: 0.9204 - val_loss: 0.2800
Epoch 8/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 34s 253ms/step - accuracy: 0.9627 - loss: 0.1482 - val_accuracy: 0.9079 - val_loss: 0.3071
Epoch 9/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 34s 253ms/step - accuracy: 0.9663 - loss: 0.1332 - val_accuracy: 0.9245 - val_loss: 0.2746
Epoch 10/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 34s 254ms/step - accuracy: 0.9743 - loss: 0.1167 - val_accuracy: 0.9185 - val_loss: 0.2753
```

Project Explanation

Training The Model

second model

```
# TEXT BRANCH
text_input_2 = Input(shape=(768,), name="text_input")

# Dense layers instead of LSTM
x_text_2 = Dense(256, activation="relu")(text_input_2)
x_text_2 = Dropout(0.3)(x_text_2)
x_text_2 = Dense(128, activation="relu")(x_text_2)
x_text_2 = Dropout(0.3)(x_text_2)
```

Project Explanation

Training The Model

```
# IMAGE BRANCH
image_input_2 = Input(shape=(128, 128, 3), name="image_input")

x_image_2 = Conv2D(32, (3, 3), activation="relu")(image_input_2)
x_image_2 = MaxPooling2D((2, 2))(x_image_2)

x_image_2 = Conv2D(64, (3, 3), activation="relu")(x_image_2)
x_image_2 = MaxPooling2D((2, 2))(x_image_2)

x_image_2 = Conv2D(128, (3, 3), activation="relu")(x_image_2)
x_image_2 = MaxPooling2D((2, 2))(x_image_2)

# Much smaller and more stable than Flatten
x_image_2 = GlobalAveragePooling2D()(x_image_2)
x_image_2 = Dense(128, activation="relu")(x_image_2)
x_image_2 = Dropout(0.4)(x_image_2)
```

Project Explanation

Training The Model

```
# CONCATENATE
combined_2 = Concatenate()([x_text_2, x_image_2])
x_2 = Dense(128, activation="relu")(combined_2)
x_2 = Dropout(0.3)(x_2)

output_2 = Dense(4, activation="softmax")(x_2)
```

Project Explanation

Training The Model

```
# MODEL
model_2 = Model(inputs=[text_input_2, image_input_2], outputs=output_2)
model_2.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])

model_2.summary()
```

Project Explanation

Training The Model

Layer (type)	Output Shape	Param #	Connected to
image_input (InputLayer)	(None, 128, 128, 3)	0	-
conv2d_2 (Conv2D)	(None, 126, 126, 32)	896	image_input[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 63, 63, 32)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 61, 61, 64)	18,496	max_pooling2d_2[…
max_pooling2d_3 (MaxPooling2D)	(None, 30, 30, 64)	0	conv2d_3[0][0]
text_input (InputLayer)	(None, 768)	0	-
conv2d_4 (Conv2D)	(None, 28, 28, 128)	73,856	max_pooling2d_3[…
dense_2 (Dense)	(None, 256)	196,864	text_input[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 128)	0	conv2d_4[0][0]
dropout_2 (Dropout)	(None, 256)	0	dense_2[0][0]
global_average_poo… (GlobalAveragePool…	(None, 128)	0	max_pooling2d_4[…
dense_3 (Dense)	(None, 128)	32,896	dropout_2[0][0]
dense_4 (Dense)	(None, 128)	16,512	global_average_p…
dropout_3 (Dropout)	(None, 128)	0	dense_3[0][0]
dropout_4 (Dropout)	(None, 128)	0	dense_4[0][0]
concatenate_1 (Concatenate)	(None, 256)	0	dropout_3[0][0], dropout_4[0][0]
dense_5 (Dense)	(None, 128)	32,896	concatenate_1[0]…
dropout_5 (Dropout)	(None, 128)	0	dense_5[0][0]
dense_6 (Dense)	(None, 4)	516	dropout_5[0][0]

Project Explanation

Training The Model

```
# computing class weights for better generalization
from sklearn.utils.class_weight import compute_class_weight

class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(y_train), y=y_train)
class_weight_dict = dict(enumerate(class_weights))
class_weight_dict
```

```
{0: np.float64(0.8428816734311583),
 1: np.float64(1.0525341130604289),
 2: np.float64(1.476887308533917),
 3: np.float64(0.8428816734311583)}
```

Project Explanation

Training The Model

```
label_encoder.classes_
```

```
array(['Casual', 'Ethnic', 'Formal', 'Sports'], dtype=object)
```

Project Explanation

Training The Model

```
# Step 1: Convert text embeddings to numpy arrays
x_train_text_embeddings_2 = np.stack(x_train['productEmbedding'].values)
x_test_text_embeddings_2 = np.stack(x_test['productEmbedding'].values)

# Step 2: Image data
x_train_image_data_2 = np.stack(x_train['image_pixels'])
x_test_image_data_2  = np.stack(x_test['image_pixels'])

# Step 3: Train the model
history = model_2.fit(
    {'text_input': x_train_text_embeddings_2, 'image_input': x_train_image_data_2},
    y_train,
    epochs=10,
    batch_size=64,
    validation_split=0.2,
    class_weight=class_weight_dict
)
```

Project Explanation

Training The Model

```
Epoch 1/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 45s 331ms/step - accuracy: 0.9374 - loss: 0.1538 - val_accuracy: 0.9296 - val_loss: 0.2191
Epoch 2/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 46s 339ms/step - accuracy: 0.9405 - loss: 0.1494 - val_accuracy: 0.9176 - val_loss: 0.2378
Epoch 3/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 42s 308ms/step - accuracy: 0.9398 - loss: 0.1485 - val_accuracy: 0.9287 - val_loss: 0.2223
Epoch 4/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 46s 337ms/step - accuracy: 0.9448 - loss: 0.1369 - val_accuracy: 0.9208 - val_loss: 0.2388
Epoch 5/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 46s 339ms/step - accuracy: 0.9480 - loss: 0.1313 - val_accuracy: 0.9333 - val_loss: 0.2194
Epoch 6/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 46s 339ms/step - accuracy: 0.9488 - loss: 0.1272 - val_accuracy: 0.9319 - val_loss: 0.2151
Epoch 7/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 47s 346ms/step - accuracy: 0.9481 - loss: 0.1301 - val_accuracy: 0.9319 - val_loss: 0.2158
Epoch 8/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 46s 340ms/step - accuracy: 0.9492 - loss: 0.1243 - val_accuracy: 0.9194 - val_loss: 0.2400
Epoch 9/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 44s 324ms/step - accuracy: 0.9535 - loss: 0.1195 - val_accuracy: 0.9315 - val_loss: 0.2098
Epoch 10/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 45s 337ms/step - accuracy: 0.9573 - loss: 0.1089 - val_accuracy: 0.9356 - val_loss: 0.2175
```

Project Explanation

Model Evaluation

model evaluation

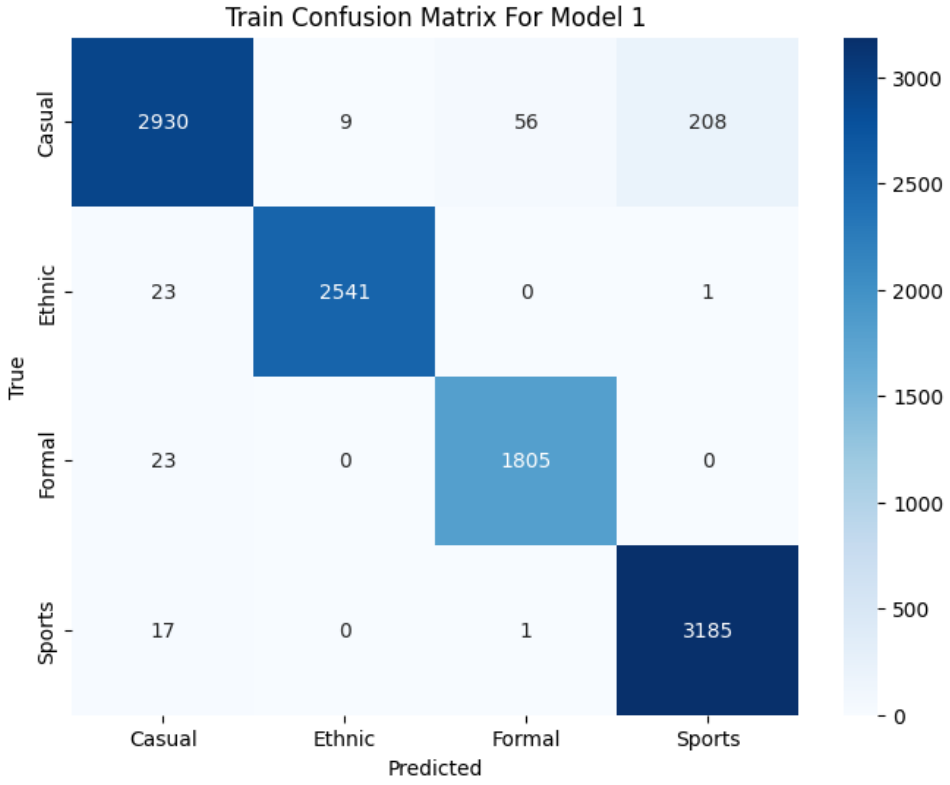
first model evaluation

```
# testing model for underfitting
y_train_pred = model_1.predict([x_train_text_embeddings_1, x_train_image_data_1])
y_train_pred_classes = np.argmax(y_train_pred, axis=1)

# printing accuracy
from sklearn.metrics import accuracy_score
print(f'train accuracy : {accuracy_score(y_train , y_train_pred_classes)}')
```

```
338/338 ━━━━━━━━━━━━━━━━━━━━ 15s 44ms/step
train accuracy : 0.9687008056301509
```

Project Explanation

Model Evaluation



Project Explanation

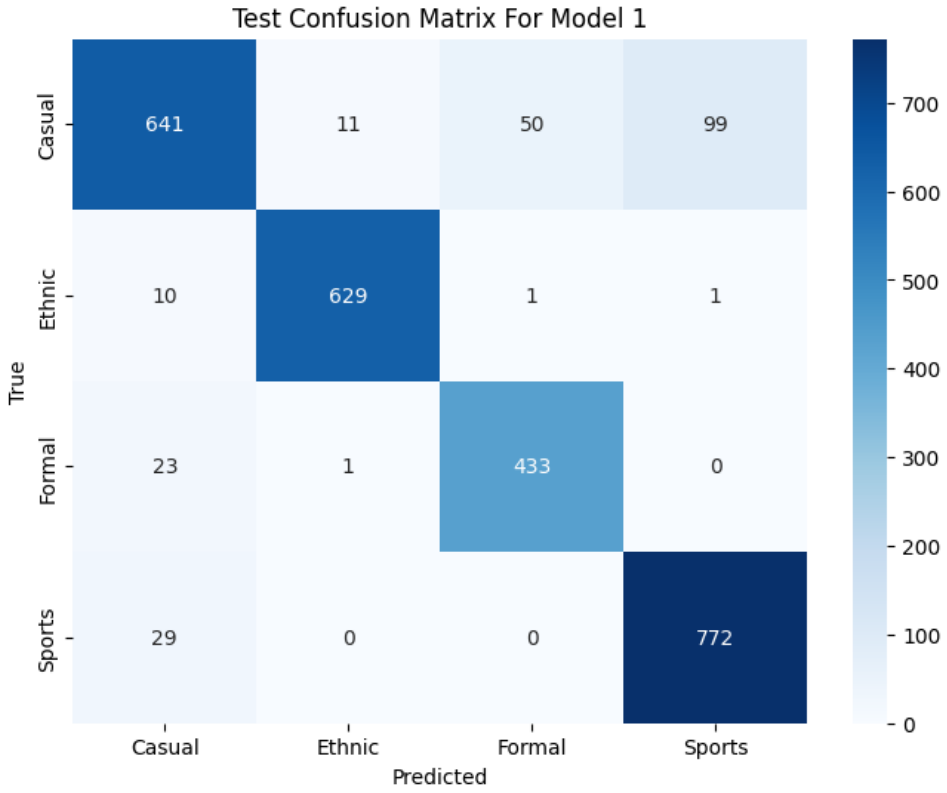
Model Evaluation

```python
# STEP 1 — stack test embeddings
x_test_text_embeddings = np.stack(x_test['productEmbedding'].values)

# reshape to (batch, 1, 768)
x_test_text_embeddings = x_test_text_embeddings.reshape((-1, 1, 768))

# STEP 2 — images
x_test_image_data = np.stack(x_test['image_pixels'])

# STEP 3 — prediction
y_test_pred = model_1.predict({
    'text_input': x_test_text_embeddings,
    'image_input': x_test_image_data
})

# STEP 4 — accuracy
y_test_pred_classes = np.argmax(y_test_pred, axis=1)
print(f"Test accuracy: {accuracy_score(y_test, y_test_pred_classes)}")
```

```
85/85 ━━━━━━━━━━━━━━━━━━━━ 4s 46ms/step
Test accuracy: 0.9166666666666666
```

Project Explanation

Model Evaluation



Project Explanation

Model Evaluation

```
# evaluating using f1 score on train and test
from sklearn.metrics import f1_score
m1_train = f1_score(y_train , y_train_pred_classes , average="weighted")
m1_test = f1_score(y_test , y_test_pred_classes , average="weighted")
print(f'train f1 score : {m1_train}')
print(f'test f1 score : {m1_test}')
```

```
train f1 score : 0.9685074817448878
test f1 score : 0.915400210554608
```

Project Explanation

Model Evaluation

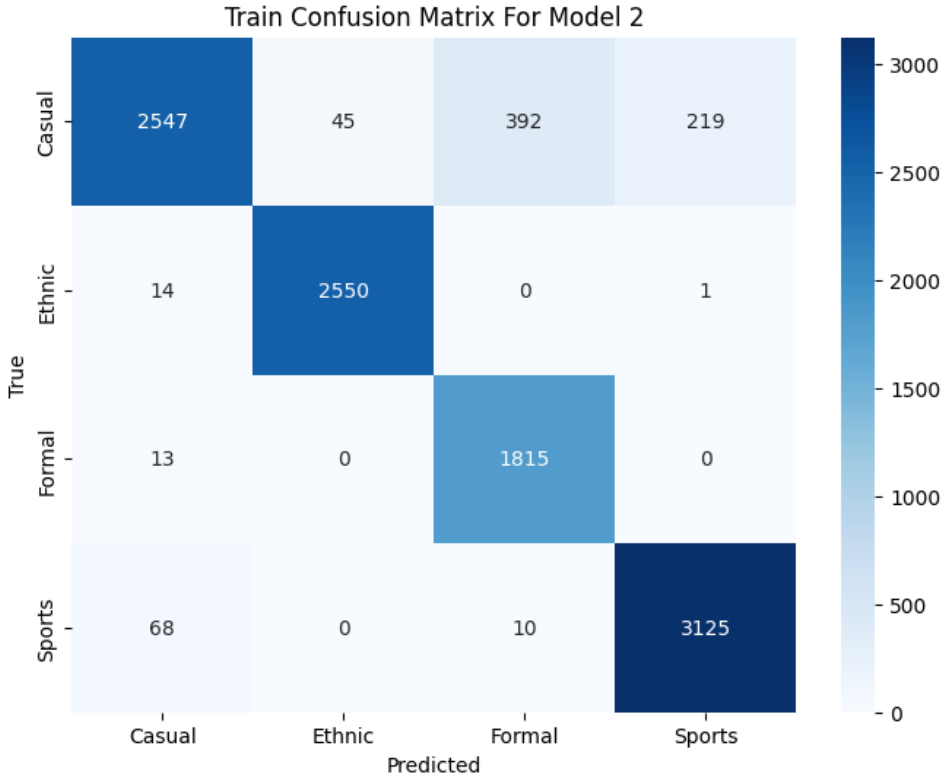
Second Model Evaluation

```
# testing model for underfitting
y_train_pred = model_2.predict([x_train_text_embeddings_2, x_train_image_data_2])
y_train_pred_classes = np.argmax(y_train_pred, axis=1)

# printing accuracy
from sklearn.metrics import accuracy_score
print(f'train accuracy : {accuracy_score(y_train , y_train_pred_classes)}')
```

```
338/338 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step
train accuracy : 0.9294379109176776
```

Project Explanation

Model Evaluation



Project Explanation

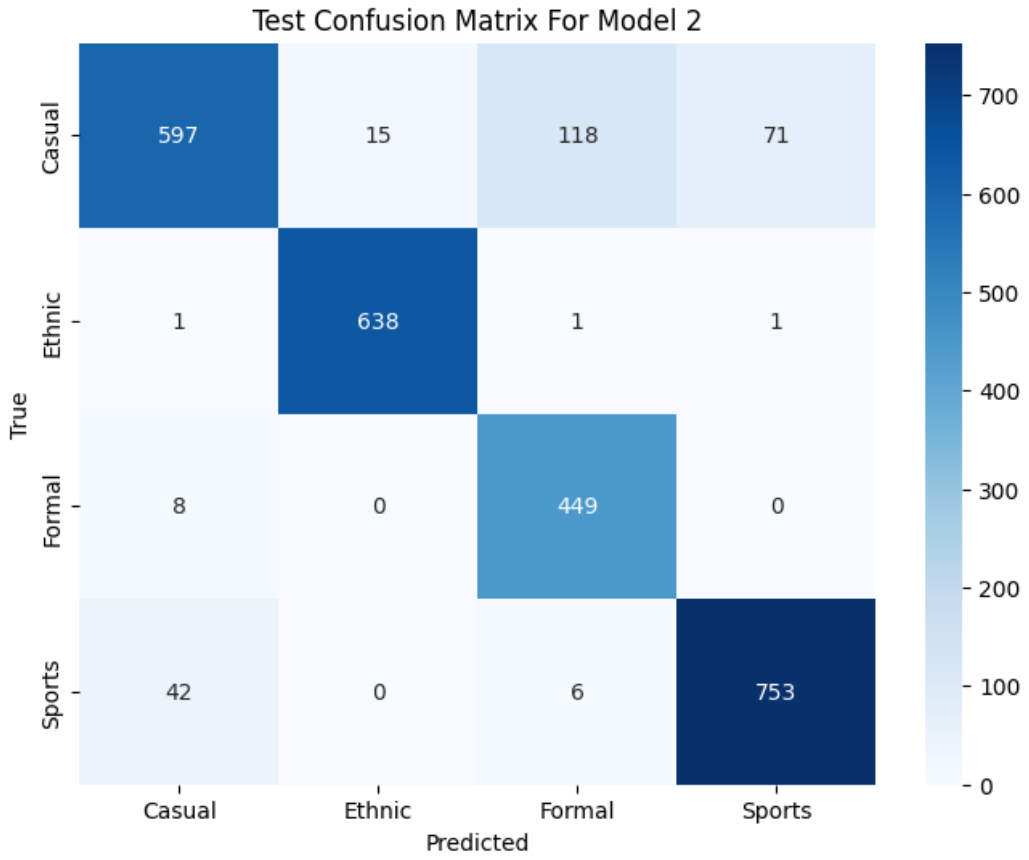
Model Evaluation

```
# STEP 1 — stack test embeddings (no reshape)
x_test_text_embeddings = np.stack(x_test['productEmbedding'].values)  # (test_size, 768)

# STEP 2 — images
x_test_image_data = np.stack(x_test['image_pixels'])                  # (test_size, 128,128,3)

# STEP 3 — prediction
y_test_pred = model_2.predict({
    'text_input': x_test_text_embeddings,
    'image_input': x_test_image_data
})

# STEP 4 — accuracy
y_test_pred_classes = np.argmax(y_test_pred, axis=1)

print(f"Test accuracy: {accuracy_score(y_test, y_test_pred_classes)}")
```

```
85/85 ━━━━━━━━━━━━━━━━━━━━ 4s 50ms/step
Test accuracy: 0.9025925925925926
```

Project Explanation

Model Evaluation



Project Explanation

Model Evaluation

```
# evaluating using f1 score on train and test
from sklearn.metrics import f1_score
m2_train = f1_score(y_train , y_train_pred_classes , average="weighted")
m2_test = f1_score(y_test , y_test_pred_classes , average="weighted")
print(f'train f1 score : {m2_train}')
print(f'test f1 score : {m2_test}')
```

```
train f1 score : 0.9282637490478018
test f1 score : 0.9007603314114011
```

Project Explanation

Model Evaluation

Comparing The Models

```
# evaluating using f1 score on train and test
from sklearn.metrics import f1_score
print(f'Model 1 train f1 score : {m1_train}')
print(f'Model 1 test f1 score : {m1_test}')
print("-"*50)
print(f'Model 2 train f1 score : {m2_train}')
print(f'Model 2 test f1 score : {m2_test}')
```

```
Model 1 train f1 score : 0.9685074817448878
Model 1 test f1 score : 0.915400210554608
--------------------------------------------------
Model 2 train f1 score : 0.9282637490478018
Model 2 test f1 score : 0.9007603314114011
```

> model 2 is less overfitting than model 1

Project Explanation

Deploying The Model

```
    # saving model
    model_2.save('model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`.
```

Project Explanation

Deploying The Model

```
from flask import Flask, request, jsonify
import numpy as np
from PIL import Image
import tensorflow as tf
from transformers import AutoTokenizer, AutoModel
import torch
import cv2
import joblib
import traceback
```

Project Explanation

Deploying The Model

```
app = Flask(__name__)

# ========================================================
#                   LOAD MODELS
# ========================================================
model = tf.keras.models.load_model("model.h5")

# Load Label Encoder
label_encoder = joblib.load("label_encoder.joblib")
```

Project Explanation

Deploying The Model

```
# ====================================================
#                    LOAD BERT
# ====================================================
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
bert_model = AutoModel.from_pretrained("bert-base-uncased")

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
bert_model.to(device)
bert_model.eval()
```

Project Explanation

Deploying The Model

```python
# ====================================================
#                 IMAGE PREPROCESSING
# ====================================================
def preprocess_image_api(img):

    # Ensure numpy array (already BGR)
    img = np.array(img)

    # If grayscale
    if img.ndim == 2:
        img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)

    # If RGBA
    if img.shape[2] == 4:
        img = cv2.cvtColor(img, cv2.COLOR_BGRA2BGR)

    # Sharpen
    kernel = np.array([[0, -1, 0],
                       [-1, 5, -1],
                       [0, -1, 0]])
    img = cv2.filter2D(img, -1, kernel)

    # Resize
    img = cv2.resize(img, (128, 128), interpolation=cv2.INTER_AREA)

    # Normalize
    img = img.astype("float32") / 255.0

    return np.expand_dims(img, axis=0)
```

Project Explanation

Deploying The Model

```python
# ====================================================
#                TEXT EMBEDDING
# ====================================================
def embed_text_api(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True).to(device)

    with torch.no_grad():
        outputs = bert_model(**inputs)

    vec = outputs.last_hidden_state.mean(dim=1).squeeze().cpu().numpy()

    return np.expand_dims(vec.astype("float32"), axis=0)
```

Project Explanation

Deploying The Model

```
# ========================================================
#                MULTIMODAL
# ========================================================
@app.route("/predict_multimodal", methods=["POST"])
def predict_multimodal():
    try:
        # --------- IMAGE PART ---------
        if "file" not in request.files:
            return jsonify({"error": "No image uploaded"}), 400

        file = request.files["file"]
        file_bytes = np.frombuffer(file.read(), np.uint8)
        img = cv2.imdecode(file_bytes, cv2.IMREAD_COLOR)
        image_input = preprocess_image_api(img)

        # --------- TEXT PART ---------
        text = request.form.get("text", "")
        if len(text.strip()) == 0:
            return jsonify({"error": "Text is empty"}), 400

        text_vec = embed_text_api(text)
```

Project Explanation

Deploying The Model

```
    # --------- PREDICT ---------
    preds = model.predict([text_vec, image_input])

    cls = int(np.argmax(preds))
    probs = preds[0].tolist()
    label = label_encoder.inverse_transform([cls])[0]

    return jsonify({
        "prediction": cls,
        "label": label,
        "probabilities": probs
    })

except Exception as e:
    traceback.print_exc()
    return jsonify({"error": str(e)}), 500
```

Project Explanation

Deploying The Model

```
# ================================================
#                 RUN SERVER
# ================================================
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, debug=True)
```

Project Explanation

Deploying The Model

```
import streamlit as st
import requests
from PIL import Image
import io
```

Project Explanation

Deploying The Model

```
API_URL = "http://192.168.1.13:5000"   # Flask API URL
LABELS = ["Casual", "Ethnic", "Formal", "Sports"]

bg_url = "https://thumbs.dreamstime.com/b/people-shopping-clothes-store-retail-scene-vector-design-generat
# Set background image
st.markdown(
    f"""
    <style>
    .stApp {{
        background: url("{bg_url}");
        background-size: cover;
        background-position: center;
        background-repeat: no-repeat;
    }}
    </style>
    """,
    unsafe_allow_html=True
```

Project Explanation

Deploying The Model

```python
st.title("Multimodal Classifier (Image + Text)")

def show_probability_bars(probs, labels):
    import streamlit as st

    st.subheader("Class Probabilities")

    for label, p in zip(labels, probs):
        st.write(f"**{label} — {p*100:.2f}%**")
        st.progress(float(p))
```

Project Explanation

Deploying The Model

```
# ================================
#   MULTIMODAL PREDICTION
# ================================

multi_image = st.file_uploader("Upload image for multimodal", type=["jpg","jpeg","png"])
multi_text = st.text_area("Enter description text for multimodal")

if st.button("Predict"):
    if not multi_image:
        st.error("Please upload an image.")
    elif not multi_text.strip():
        st.error("Please enter text.")
    else:
        try:
            files = {
                "file": (multi_image.name, multi_image.getvalue(), multi_image.type)
            }
            data = {"text": multi_text}

            response = requests.post(
                f"{API_URL}/predict_multimodal",
                files=files,
                data=data
            )
```

Project Explanation

Deploying The Model

```
        result = response.json()
        pred_idx = result["prediction"]

        st.success(f"Final Prediction: **{LABELS[pred_idx]}**")
        show_probability_bars(result["probabilities"], LABELS)

    except Exception as e:
        st.error("Error contacting multimodal API.")
        st.error(str(e))
```

Model Features

1. Supports both **image data** and **text descriptions**
2. Uses CNNs to extract detailed visual features.
3. Uses text encoders for captions or product descriptions.
4. Accepts user-uploaded images for instant classification.
5. Lightweight API for integration with mobile/web apps.
6. Fast inference time.

Future integration

The model have a lot of potential in integration with online shopping as it can be integrated with cloth selling websites to can help the customers by suggesting different piece of clothes or making up new outfits for them based an their needs