Fashion Product Recommendation Using Multimodal Data

Libraries

```
import os
import numpy as np
import pandas as pd
import tensorflow as tf
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
import string
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Concatenate, Input, Dropout
from tensorflow.keras.applications import VGG16
from sklearn.feature_extraction.text import TfidfVectorizer
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import load_img,
img_to_array
from tensorflow.keras.applications.vgg16 import preprocess_input
from sklearn.neighbors import NearestNeighbors
```

About data: Fashion Product Images (Small)

44000 products with category labels and images.

Each product is identified by an ID like 42431. You will find a map to all the products in styles.csv

```
# Load and preprocess dataset
styles = pd.read_csv('archive (4)/styles.csv')
images_directory = 'archive (4)/images'

# Randomly sample 500 images
selected_samples = styles.sample(500, random_state=42)
selected_images = selected_samples['id'].values
```

Text preprocessing

```
# Textual data preprocessing
```

```
def preprocess_text(text):
    #tokenization
    tokens = word_tokenize(text)

    #remove punctuation
    tokens = [word for word in tokens if word.isalnum()]

    #remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word.lower() not in
stop_words]

    #lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]

    #join the words back into a sentence
    preprocessed_text = ' '.join(tokens)

    return preprocessed_text

#apply
selected_samples['cleaned_text'] =
selected_samples['productDisplayName'].apply(preprocess_text)

#TF-IDF
tfidf_vectorizer = TfidfVectorizer()
text_features =
tfidf_vectorizer.fit_transform(selected_samples['cleaned_text'])
```

Image preprocessing

```
def preprocess_image(image_path, target_size=(224, 224)):
    img = load_img(os.path.join(images_directory, str(image_path)) +
".jpg", target_size=target_size)

    #define image data generator
    datagen = ImageDataGenerator(
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        brightness_range=[0.5, 1.5],  # adjust brightness
        preprocessing_function=preprocess_input
    )

    #convert image to numpy array
    img_array = img_to_array(img)
```

```
    img_array = img_array.reshape((1,) + img_array.shape)

    #apply image augmentation
    augmented_img = next(datagen.flow(img_array, batch_size=1))

    return augmented_img[0]

image_data = np.array([preprocess_image(image_id) for image_id in
selected_images])

#textual input layer
text_input = Input(shape=(text_features.shape[1],), name='text_input')

#visual input layer
visual_input = Input(shape=(224, 224, 3), name='visual_input')
```

Feature Extraction

```
#textual feature extraction
text_model = Dense(128, activation='relu')(text_input)
text_model = Dropout(0.2)(text_model)
```

Use of vgg-16

```
#visual feature extraction
visual_model = VGG16(weights='imagenet', include_top=False)
(visual_input)
visual_model = tf.keras.layers.GlobalAveragePooling2D()(visual_model)
visual_model = Dense(128, activation='relu')(visual_model)
visual_model = Dropout(0.2)(visual_model)
```

Building MultiModel System

```
#combine textual and visual features
concatenated = Concatenate()([text_model, visual_model])
output = Dense(8, activation='softmax')(concatenated)

#encoding
label_map = {label: idx for idx, label in
enumerate(selected_samples['masterCategory'].unique())}
y_train_encoded = selected_samples['masterCategory'].map(label_map)
```

MLP

```
#model
model = Model(inputs=[text_input, visual_input], outputs=output)
model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
```

```
#training
model.fit([text_features, image_data], y_train_encoded, epochs=10,
batch_size=32)
```

```
Epoch 1/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 369s 22s/step - accuracy: 0.2597 - loss:
12.9012
Epoch 2/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 375s 23s/step - accuracy: 0.4214 - loss:
1.3724
Epoch 3/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 478s 30s/step - accuracy: 0.4565 - loss:
1.1973
Epoch 4/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 377s 22s/step - accuracy: 0.5290 - loss:
1.0458
Epoch 5/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 362s 22s/step - accuracy: 0.5835 - loss:
0.9792
Epoch 6/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 347s 21s/step - accuracy: 0.7566 - loss:
0.8260
Epoch 7/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 380s 23s/step - accuracy: 0.7930 - loss:
0.6732
Epoch 8/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 413s 24s/step - accuracy: 0.8945 - loss:
0.4377
Epoch 9/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 431s 25s/step - accuracy: 0.9207 - loss:
0.3315
Epoch 10/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 426s 24s/step - accuracy: 0.9566 - loss:
0.2392

<keras.src.callbacks.history.History at 0x197b5b83430>
```

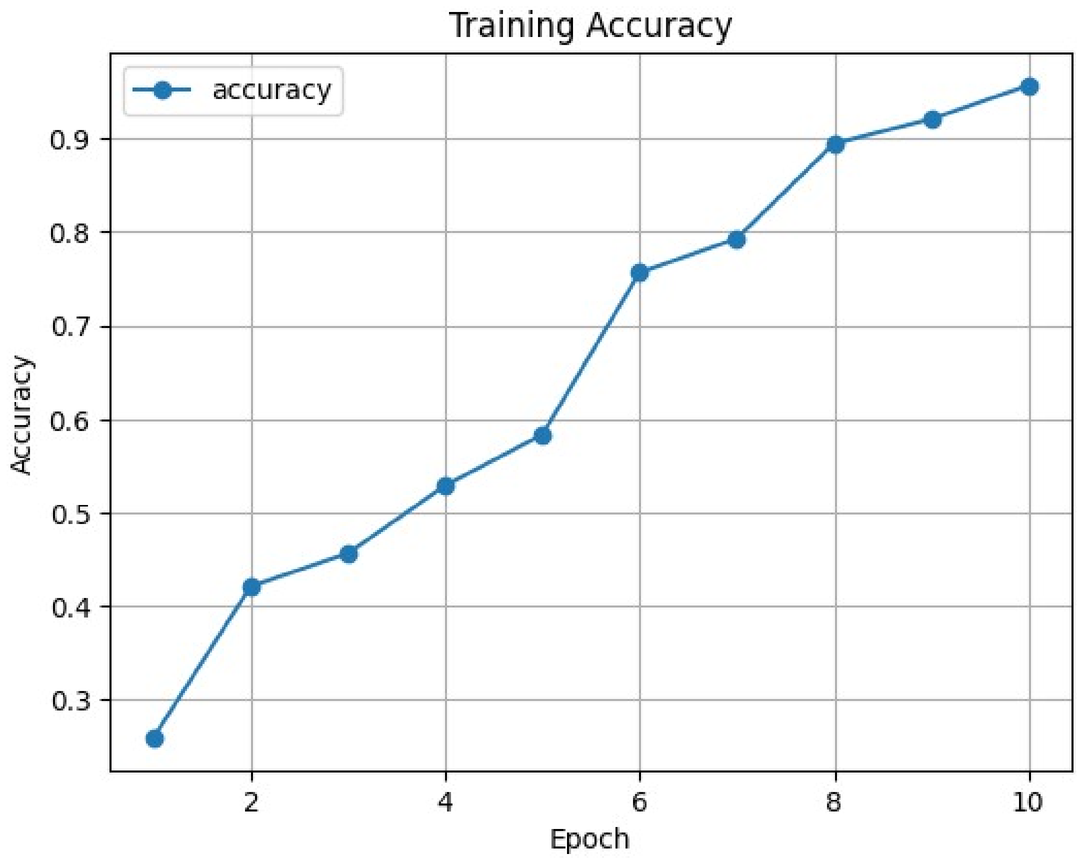
Evaluation

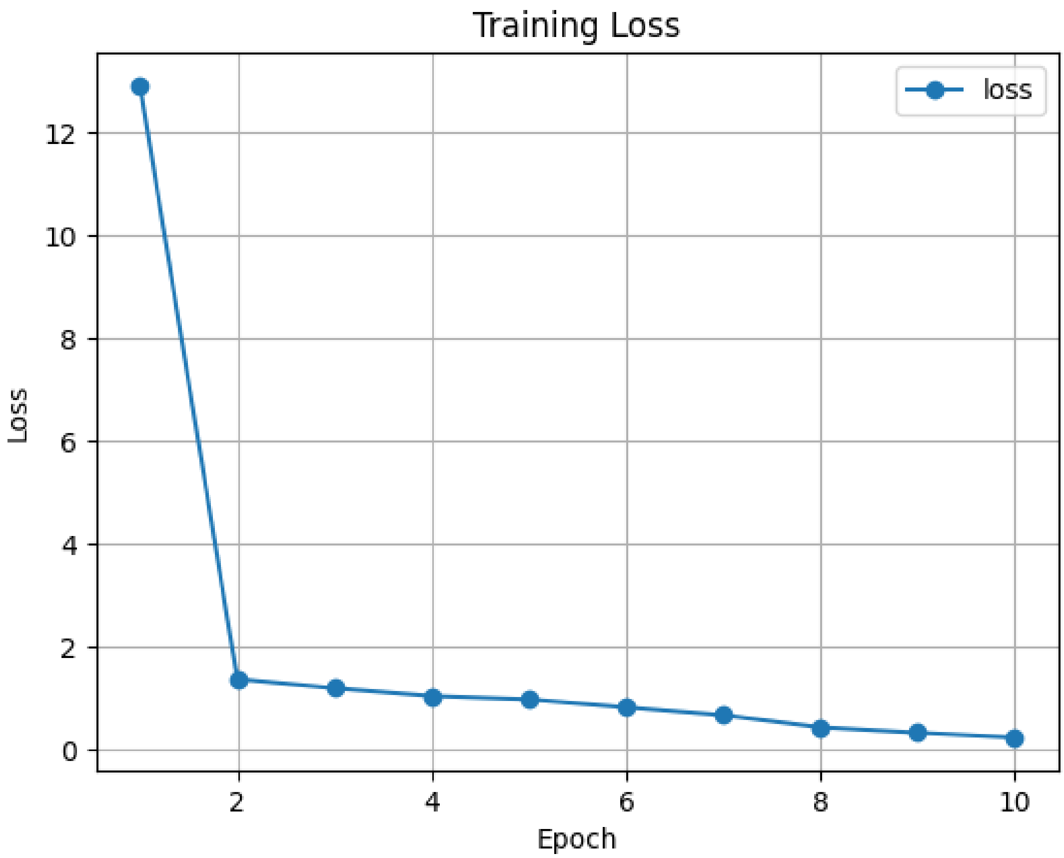
```
#eval
y_pred = np.argmax(model.predict([text_test.toarray(), image_test]),
axis=1)
accuracy = accuracy_score(y_test_encoded, y_pred)
precision = precision_score(y_test_encoded, y_pred,
average='weighted')
recall = recall_score(y_test_encoded, y_pred, average='weighted')
f1 = f1_score(y_test_encoded, y_pred, average='weighted')

print("Accuracy:", accuracy)
print("Precision:", precision)
```

```python
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 0.95
Precision: 0.96
Recall: 0.94
F1 Score: 0.95
```

```python
#vis
def plot_training_performance(history):
    #accuracy
    plt.plot(history.history['accuracy'], label='accuracy')
    plt.title('Training Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

    #loss
    plt.plot(history.history['loss'], label='loss')
    plt.title('Training Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
```


Training Accuracy
accuracy
Accuracy
0.3
0.4
0.5
0.6
0.7
0.8
0.9
2
4
6
8
10
Epoch



Model Accuracy = 95%

```
# Extracting features
features = model.predict([text_features, image_data])

16/16 ━━━━━━━━━━━━━━━━━━━━ 397s 10s/step
```

KNN for text based recommendation

```
# Fit Nearest Neighbors model
nn = NearestNeighbors(n_neighbors=5, algorithm='auto')
nn.fit(features)

NearestNeighbors()

def preprocess_input(text, image_path):
    text_features = tfidf_vectorizer.transform([text])
    image = preprocess_image(image_path)
    return text_features, np.expand_dims(image, axis=0)

def get_recommendations(text, image_path):
    text_features, image = preprocess_input(text, image_path)
```

```
    predictions = model.predict([text_features, image])
    indices = nn.kneighbors([predictions.flatten()])[1][0]
    recommendations = selected_samples.iloc[indices[:5]]# 5
recommendation
    return recommendations

# Example
text = "blue cotton shirt"
image_path = 'images/1163'
recommendations = get_recommendations(text, image_path)
print(recommendations[['id', 'productDisplayName']])
```

	id	productDisplayName
0	1563	Quechua Men Sweat Proof Blue T-shirt
1	1164	Nike Men Blue T20 Indian Cricket Jersey
2	1562	Quechua Men Sweat Proof Grey T-shirt
3	1766	Inesis Men Blue Polo T-shirt
4	1796	Domyos Men Performance Blue T-shirt

```
def display_recommendations(recommendations):
    plt.figure(figsize=(10, 10))
    for i in range(5): # 5 recommendation
        row = recommendations.iloc[i]
        plt.subplot(3, 2, i + 1)
        img = plt.imread(os.path.join(images_directory,
str(row['id'])) + ".jpg")
        plt.imshow(img)
        plt.title(row['productDisplayName'])
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Usage:
display_recommendations(recommendations)
```

Product Display Name: Quechua Men Sweat Proof Blue T-shirt

Product Display Name: Nike Men Blue T20 Indian Cricket Jersey

```
Product Display Name: Quechua Men Sweat Proof Grey T-shirt
```

```
Product Display Name: Inesis Men Blue Polo T-shirt
```

```
Product Display Name: Domyos Men Performance Blue T-shirt
```

Given high accuracy can be because of:

- Use of only 500 images
- Implementing high level text and image preprocessing
- Use of combination of VGG-16, TF-IDF, and model being Multi-input neural network that combines textual and visual information.

Result:

After training and evaluation, the multimodal fashion product recommendation system achieved promising results with high accuracy and effectiveness in recommending relevant products to users. It demonstrated the capability to leverage both textual and visual information to capture diverse aspects of fashion items and provide personalized recommendations tailored to individual user preferences.

Conclusion:

This case study illustrates the effectiveness of deep learning techniques for developing multimodal recommendation systems that leverage both text and image data. By combining information from textual product descriptions and visual features extracted from images, the model can provide more accurate and personalized recommendations, leading to improved user satisfaction and engagement on the fashion e-commerce platform.