FINAL PROJECT – D2

SVM POWERED ART MOVEMENT CLASSIFIER

SOURCE CODE

ROHAN GAMIDI AIE22019

SNEHA SARAGADAM AIE22057

VINITHA CHOWDARY A AIE22066

```
import pandas as pd
from sklearn import svm
from sklearn.model_selection import GridSearchCV
import os
import matplotlib.pyplot as plt
import seaborn as sns
from skimage.transform import resize
from skimage.io import imread
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import pickle
import tkinter as tk
from tkinter import filedialog
from PIL import Image, ImageTk
import random

Categories = ['Japanese_Art', 'Expressionism', 'Primitivism']
```

```
flat_data_arr = []
target_arr = []

datadir = r"D:\College\Sem_3\MFC\Final Project\Art Style Dataset"

for i in Categories:
    print(f'loading... category : {i}')
    path = os.path.join(datadir, i)
    for img in os.listdir(path):
        img_array = imread(os.path.join(path, img))
        img_resized = resize(img_array, (150, 150, 3))
        flat_data_arr.append(img_resized.flatten())
        target_arr.append(Categories.index(i))
    print(f'loaded category:{i} successfully')

flat_data = np.array(flat_data_arr)
target = np.array(target_arr)
df = pd.DataFrame(flat_data)
df['Target'] = target

# Splitting the data into training and testing data
x = df.iloc[:, :-1]
y = df.iloc[:, -1]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state=77,
stratify=y)
print('Splitted Successfully')

# Check if the model pickle file already exists
model_file_path = 'img_model.p'
if os.path.exists(model_file_path):
```

```
    # Load the trained model
    model = pickle.load(open(model_file_path, 'rb'))
else:
    # Training the model
    param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [0.0001, 0.001, 0.1, 1], 'kernel': ['rbf', 'poly']}
    svc = svm.SVC(probability=True)
    print("The training of the model is started, please wait for a while as it may take a few minutes to complete")
    model = GridSearchCV(svc, param_grid)
    model.fit(x_train, y_train)
    print('The Model is trained well with the given images')
    model.best_params_

    # Save the model using Pickle
    pickle.dump(model, open(model_file_path, 'wb'))
    print("Pickle is dumped successfully")

# Create a tkinter GUI
root = tk.Tk()
root.title("Art Style Classifier")

# Function to classify the uploaded image
def classify_image():
    file_path = filedialog.askopenfilename(title="Select Image", filetypes=[("Image files", "*.jpg;*.jpeg;*.png")])
    if file_path:
        img = imread(file_path)
        img_resize = resize(img, (150, 150, 3))
        img_flatten = img_resize.flatten()
        img_df = pd.DataFrame([img_flatten])
```

```
        # Predictions on test set
        y_pred_test = model.predict(x_test)

        # Confusion matrix on test set
        confusion_mat = confusion_matrix(y_test, y_pred_test)
        print("Confusion Matrix on Test Set:")
        print(confusion_mat)

        predicted_label = model.predict(img_df)
        accuracy = model.predict_proba(img_df)
        result_label.config(text=f"Predicted art style: {Categories[predicted_label[0]]}")
        accuracy_label.config(text=f"Accuracy: {accuracy[0][predicted_label[0]]*100:.2f}%")

        # Display the confusion matrix
        plt.figure(figsize=(8, 6))
        sns.heatmap(confusion_mat, annot=True, fmt="d", cmap="Blues",
xticklabels=Categories, yticklabels=Categories)
        plt.title('Confusion Matrix')
        plt.xlabel('Predicted Label')
        plt.ylabel('True Label')
        plt.show()

# Function to recommend images based on the classified text
def recommend_images():
    predicted_style = result_label.cget("text").replace("Predicted art style: ", "")
    if predicted_style:
        # Path to the folder containing images for the predicted art style
        predicted_style_folder = os.path.join(datadir, predicted_style)
        # List all files in the predicted art style folder
```

```
        all_files = os.listdir(predicted_style_folder)
        # Select 5 random files (if available)
        selected_files = random.sample(all_files, min(5, len(all_files)))
        # Display the recommended images
        display_recommended_images(predicted_style_folder, selected_files)
    else:
        recommendation_label.config(text="Error: Predicted art style not available.")

# Function to display recommended images
def display_recommended_images(folder_path, selected_files):
    recommendation_window = tk.Toplevel()
    recommendation_window.title("Recommended Images")
    for file_name in selected_files:
        file_path = os.path.join(folder_path, file_name)
        img = Image.open(file_path)
        img = img.resize((100, 100), resample=Image.ANTIALIAS if hasattr(Image, 'ANTIALIAS')
else 3)  # Use 3 as a fallback
        img_tk = ImageTk.PhotoImage(img)
        image_label = tk.Label(recommendation_window, image=img_tk)
        image_label.image = img_tk
        image_label.pack(side=tk.LEFT, padx=5, pady=5)
    recommendation_window.after(5000, recommendation_window.destroy)



# Create GUI components
classify_button = tk.Button(root, text="Classify Image", command=classify_image)
classify_button.pack(pady=10)


recommend_button = tk.Button(root, text="Recommend Images",
command=recommend_images)
```

```
recommend_button.pack(pady=10)

result_label = tk.Label(root, text="", font=("Helvetica", 12))
result_label.pack(pady=10)

accuracy_label = tk.Label(root, text="", font=("Helvetica", 12))
accuracy_label.pack(pady=10)

recommendation_label = tk.Label(root, text="", font=("Helvetica", 12))
recommendation_label.pack(pady=10)

# Start the tkinter event loop
root.mainloop()
```