Text to gesture conversion

```
import os
from tkinter import Tk, Label, Button, Entry, font
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model
import numpy as np
from PIL import Image, ImageTk

class CustomResNetModel:
    def __init__(self, model_path='resnet50_custom_model.h5', image_size=(224, 224)):
        # Load the trained model
        self.model = load_model(model_path)
        self.image_size = image_size

    def preprocess_image(self, image_path):
        # Load and preprocess the image
        img = image.load_img(image_path, target_size=self.image_size)
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array /= 255.0  # Rescale to match the preprocessing in the training data generator
        return img_array

    def predict_class(self, image_path):
        # Preprocess the image
        img_array = self.preprocess_image(image_path)

        # Make predictions
        predictions = self.model.predict(img_array)
```

```
        # Get the class index with the highest probability
        predicted_class_index = np.argmax(predictions)

        # Map the class index to the actual class label
        class_labels = list(self.model.layers[-1].get_config()['class_name_map'].values())
        predicted_class_label = class_labels[predicted_class_index]

        return predicted_class_label

# Example usage
custom_model = CustomResNetModel()
class ImageDisplayApp:
    def __init__(self, root):
        self.root = root
        self.root.title("TEXT TO SIGN RECOGNITION")
        self.root.geometry("600x600")  # Set the window size to 800x800 pixels

        self.folder_path = ""
        self.images = []
        self.current_image_index = 0

        self.label = Label(root, text="Type a text for sign prediction")
        self.label.pack()

        self.folder_entry = Entry(root)
        self.folder_entry.pack()
        button_font = font.Font(weight="bold", size=12)
        self.display_button = Button(root, text="predict sign", command=self.display_images)
        self.display_button.pack()
```

```
def display_images(self):
    self.folder_path = self.folder_entry.get()

    if not self.folder_path:
        self.label.config(text="Please type a correct text name.")
        return

    self.images = self.load_images_from_folder(self.folder_path)

    if not self.images:
        self.label.config(text=f"No images found for this text: {self.folder_path}")
        return

    # Display the first image
    self.show_image()

def load_images_from_folder(self, folder):
    image_list = []
    for filename in os.listdir(folder):
        if filename.lower().endswith(('.png', '.jpg', '.jpeg', '.gif', '.bmp')):
            image_path = os.path.join(folder, filename)
            image_list.append(image_path)
    return image_list

def show_image(self):
    if not self.images:
        return

    # Remove any previous image displayed
    for widget in self.root.winfo_children():
```

```
            if isinstance(widget, Label):
                widget.destroy()

        # Display the current image
        img_path = self.images[self.current_image_index]
        img = Image.open(img_path)
        img = img.resize((450, 450), Image.ANTIALIAS)  # Resize the image to 500x500 pixels
        img = ImageTk.PhotoImage(img)

        # Calculate the center position for the label
        x_center = (self.root.winfo_reqwidth() - img.width()) // 2
        y_center = (self.root.winfo_reqheight() - img.height()) // 2

        label = Label(self.root, image=img)
        label.image = img
        label.place(x=x_center, y=y_center)  # Use place() to set the position
        label.pack()

        # Increment the current image index
        self.current_image_index = (self.current_image_index + 1) % len(self.images)

if __name__ == "__main__":
    root = Tk()
    app = ImageDisplayApp(root)
    root.mainloop()
```

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define constants
batch_size = 32
image_size = (224, 224)

# Create an ImageDataGenerator for data augmentation and normalization
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

# Load and prepare the training dataset using flow_from_directory
train_generator = train_datagen.flow_from_directory(
    'dataset/',
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',  # Assumes you have multiple classes
    shuffle=True
)

# Load the pre-trained ResNet50 model without the top (fully connected) layers
base_model = tf.keras.applications.ResNet50(
    include_top=False,
    weights='imagenet',
```

```
    input_shape=(224, 224, 3)
)

# Freeze the layers of the pre-trained ResNet50 model
for layer in base_model.layers:
    layer.trainable = False

# Create your custom model by adding your top layers on top of ResNet50
model = models.Sequential()
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(8, activation='softmax'))  # Adjust num_classes based on your dataset

# Compile the model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',

)

# Train the model
epochs = 10  # Adjust based on your dataset and computational resources
model.fit(
    train_generator,
    epochs=epochs,
    steps_per_epoch=train_generator.samples // batch_size,
    verbose=1
)
```

```
# Save the trained model
model.save('resnet50_custom_model.h5')
```

Gesture to text conversion

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
import os

import cv2
import HandDataCollecter
import mediapipe as mp
import numpy as np

########Initialise random forest

local_path = (os.path.dirname(os.path.realpath('__file__')))

file_name = ('a data.csv')  # file of total data
data_path = os.path.join(local_path, file_name)
print(data_path)
df = pd.read_csv(r'' + data_path)

print(df)

units_in_data = 28  # no. of units in data
```

```
titles = []
for i in range(units_in_data):
    titles.append("unit-" + str(i))
X = df[titles]
y = df['letter']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=2)
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

clf = RandomForestClassifier(n_estimators=30)  # random forest
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("1.Random Forest Accuracy")
print('Accuracy: ', metrics.accuracy_score(y_test, y_pred))
cmrf = confusion_matrix(y_test, y_pred)



print("Random Forest classification_report")
print(classification_report(y_pred, y_test, labels=None))
print("Random Forest confusion_matrix")

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
print("CONFUSION MATRIX OF RF")
print(cm)
tpr = float(cm[0][0]) / np.sum(cm[0])
fpr = float(cm[1][1]) / np.sum(cm[1])
```

```
plt.figure(figsize=(12, 12))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square=True, cmap='Blues');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Confusion Matrix of RF '
plt.title(all_sample_title, size=15);
plt.show()

clf1 = KNeighborsClassifier()  # KNN
clf1.fit(X_train, y_train)
y_pred = clf1.predict(X_test)
print("2. knn Accuracy")
print('Accuracy: ', metrics.accuracy_score(y_test, y_pred))
cmrf = confusion_matrix(y_test, y_pred)



print("knn classification_report")
print(classification_report(y_pred, y_test, labels=None))



import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
print("CONFUSION MATRIX OF knn")
print(cm)
tpr = float(cm[0][0]) / np.sum(cm[0])
fpr = float(cm[1][1]) / np.sum(cm[1])
plt.figure(figsize=(12, 12))
```

```
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square=True, cmap='Blues');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Confusion Matrix of knn '
plt.title(all_sample_title, size=15);
plt.show()

from sklearn.svm import SVC
clf2 = SVC()  # SVC
clf2.fit(X_train, y_train)
print("3.svm Accuracy")
y_pred = clf2.predict(X_test)
print('Accuracy: ', metrics.accuracy_score(y_test, y_pred))
cmsvc = confusion_matrix(y_test, y_pred)



print("svm classification_report")
print(classification_report(y_pred, y_test, labels=None))
print("svm confusion_matrix")
print(cmrf)
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

tpr = float(cm[0][0]) / np.sum(cm[0])
fpr = float(cm[1][1]) / np.sum(cm[1])
plt.figure(figsize=(12, 12))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square=True, cmap='Blues');
```

```
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Confusion Matrix of svm '
plt.title(all_sample_title, size=15);
plt.show()

# Accuracy values
accuracies = [97.6, 96.9, 96.7]
model_names = ['Random Forest', 'KNN', 'SVM']

# Plotting the line graph
plt.figure(figsize=(10, 6))
plt.plot(model_names, accuracies, marker='o', color='blue', linestyle='-', linewidth=2,
markersize=8)
plt.title('Accuracy of Different Models')
plt.ylabel('Accuracy (%)')
plt.ylim(60, 100)  # Set y-axis limit to represent accuracy percentage
plt.grid(True)
plt.show()

# Accuracy values
accuracies = [97.6, 96.9, 96.7]
model_names = ['Random Forest', 'KNN', 'SVM']

# Plotting the bar graph
plt.figure(figsize=(10, 6))
plt.bar(model_names, accuracies, color=['green', 'blue', 'red'])
plt.title('Accuracy of Different Models')
plt.ylabel('Accuracy (%)')
plt.ylim(60, 100)  # Set y-axis limit to represent accuracy percentage
plt.show()
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
import os

import cv2
import HandDataCollecter
import mediapipe as mp
import numpy as np

########Initialise random forest

local_path = (os.path.dirname(os.path.realpath('__file__')))

file_name = ('a data.csv')#file of total data
data_path = os.path.join(local_path,file_name)
print (data_path)
df = pd.read_csv(r"+data_path)

print (df)

units_in_data = 28 #no. of units in data

titles = []
for i in range(units_in_data):
    titles.append("unit-"+str(i))
X = df[titles]
y = df['letter']
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.5,random_state=2)
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
clf = RandomForestClassifier(n_estimators=30)#random forest
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
cmrf=confusion_matrix(y_test, y_pred)
print("1.Random Forest Accuracy")

print("Random Forest classification_report")
print(classification_report(y_pred, y_test, labels=None))
print("Random Forest confusion_matrix")

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("CONFUSION MATRIX OF RF")
print(cm)
tpr = float(cm[0][0])/np.sum(cm[0])
fpr = float(cm[1][1])/np.sum(cm[1])
plt.figure(figsize=(12, 12))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True, cmap = 'Blues');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Confusion Matrix of RF '
plt.title(all_sample_title, size = 15);
plt.show()
```

```
#########Begin predictions
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands

def get_prediction(image):
    with mp_hands.Hands(min_detection_confidence=0.5, min_tracking_confidence=0.5) as
hands:
        ImageData = HandDataCollecter.ImageToDistanceData(image, hands)
        DistanceData = ImageData['Distance-Data']
        image = ImageData['image']
        prediction = clf.predict([DistanceData])
        return prediction[0]



if __name__ == '__main__':
    cap = cv2.VideoCapture(0)

    SpelledWord = ""
    while cap.isOpened():
            success, image = cap.read()
            if not success:
                    print("Ignoring empty camera frame.")
                    # If loading a video, use 'break' instead of 'continue'.
                    continue

            '''ImageData = HandDataCollecter.ImageToDistanceData(image, hands)
            DistanceData = ImageData['Distance-Data']
            image = ImageData['image']

            if cv2.waitKey(1) & 0xFF == 32:
                prediction = clf.predict([DistanceData])
```

```
            SpelledWord = str(prediction[0])
            #print(SpelledWord)'''
        try:
            SpelledWord = get_prediction(image)
            #cv2.putText(image,  SpelledWord, (50,50), 1, 2, 255)
            cv2.putText(image,SpelledWord,(50,50), cv2.FONT_HERSHEY_SIMPLEX, 1, (124,252,0), 5, cv2.LINE_AA)
        except:
            pass
        cv2.imshow('frame', image)
        if cv2.waitKey(5) & 0xFF == 27: #press escape to break
                break
    cap.release()
    cv2.destroyAllWindows()
```