

Handwriting Recognition

Handwriting Recognition Using Support Vector Machines and Deep Learning

Manikandan Ramalingam, University of San Diego

Submitted to: Prof. David Friesen

Submitted Date: 4/14/2024.

ABSTRACT

The purpose of this report is to find the identity of a person given his handwriting. In real-world, there are many situations where there is a need to verify the signature or one's handwriting. Some examples are verifying handwritten will, signatures in check etc. or for identifying criminals. For simplicity, only English language is considered.

Dataset Characteristics

The dataset is the set of handwritten image files (in .png format) each containing multiple sentences in English language.

Subject Area

The data files are random [1] essays picked up from the internet. Same essays were written by 2 different people.

Associated Tasks

The primary task associated with this dataset is classification, specifically binary classification to predict the identity of a person given his handwriting. Used SVM and CNN classifiers and analyzed the handwriting of 2 individuals. The output variable is just the label of the individual names either "Mani" or "Priya". This is considered as a Supervised Learning task in machine learning because the input word image files are labeled with either of the 2 names. The output would be either of these 2 labels.

Handwriting Recognition

Data Volume

The dataset comprises of 17 image files with multiple sentences. It was split into 1726-word images (.png) consisting of individual words.

Team Structure

Members	Contributions	Comments
Manikandan Ramalingam (Lead)	GitHub ReadMe, Coding and Documentation for Data collection, Data transformation, Coding and Documentation for SVM classifier, Model fit with various word images, train the model and predicting the results, CNN model fit with training and test sets, predicting the results, Analysis of Accuracy for both the models.	

Data Analysis

Data Preparation

Two random [1] essays were collected from the internet. One on William Bronk, a poet, and another on Salem a city in Tamilnadu, India. These essays were written in their own handwriting by 2 individuals (Manikandan Ramalingam and Priya Pattanam Thulasiraman). Then images were captured as .png files. Each .png file would have multiple handwritten sentences.

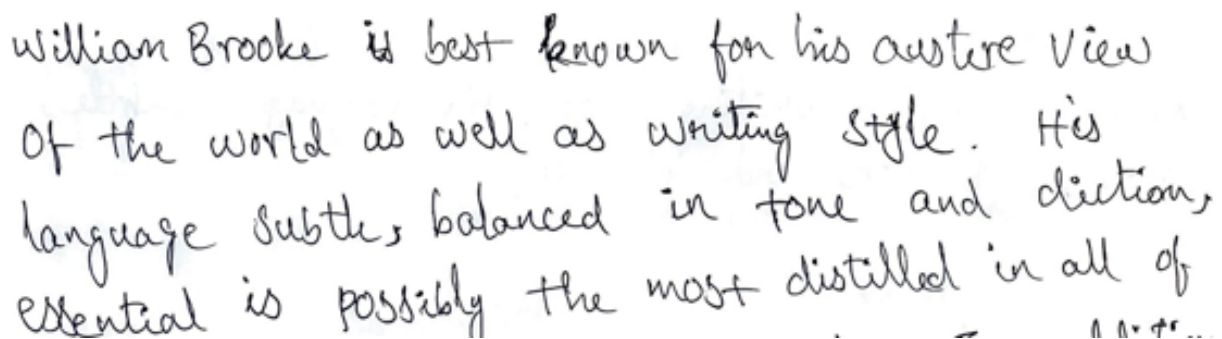
Data Transformation

Each image file is transformed into data using Optical Character Recognition (OCR) technique. The pytesseract and PIL libraries of Python were used. The output of this is text bounding boxes with each word as an image. These individual word images are saved in a particular output directory with the names of individuals written the sentences. For example, Mani_1.png, Mani_2.png or Priya_1.png, Priya_2.png etc. Here, the labels “Mani” and “Priya” will be the output predicted variable once the model gets fitted with SVM and Convolutional Neural Networks classifiers given the input as word image.

Data Visualization

The image files showing sample of full sentences and the cropped word files is shown below.

Image 1- Sentences



William Brooke is best known for his austere view of the world as well as writing style. His language subtle, balanced in tone and diction, essential is possibly the most distilled in all of

Image 2- Individual Words

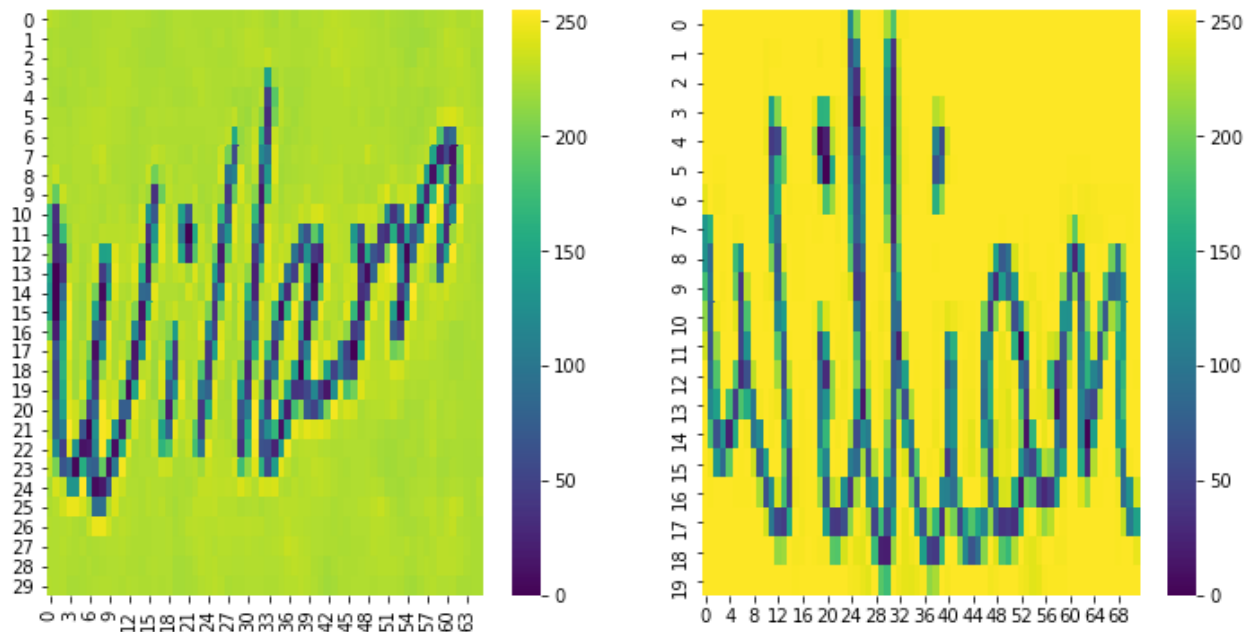
Priya_1.png sample

William

Mani_1.png sample

William

Image 3 – With Seaborn heatmap and matplotlib



Data Pre-processing

Re-shape Data

The output word image data if processed would show a 3 dimensional array (height, width and channels for RGB) of different sizes. This has to be flattened to 1-D array so that we get a single long vector of pixels to simplify the processing of images. This is also useful for the classifiers to learn patterns directly from pixel values. The data is also split manually here to train and test data sets with 80% of word images as train set and 20% of word images as test set. The

Handwriting Recognition

code of this is added in the Appendix section. The python reshape() method was used to flatten it.

Data Padding

Before fitting the model to the classifier the input image array if they are of different length, the model fit would not work. So, the maximum length of each input array is checked and the maximum array size is fetched. Then, all arrays that are of not that maximum size would be padded with a python pad() method in Numpy library with the size of maximum array size element. Then, it is scaled with the feature_range of 0, 1 using sklearn preprocessing library to have scaled values in the range of 0 to 1. This is done for both the training and test sets.

Data Encoding

The output variable was a string all along either 'Mani' or 'Priya'. But, it cannot be fitted into models directly as it expects a numerical encoding of output variables. So, the output variables are encoded with a python library Labelencoder which fits the value 0 for 'Mani' and 1 for 'Priya'.

SVM Classifier

Model

Support Vector Machine classifier is the machine learning algorithm primarily used for classification tasks such as image classification. Since, handwritten text words are in image format, SVM models can be used in this task. The svm python library from scikitlearn package was used for this purpose.

Svm package has Support Vector Classifier (SVC) for the creation of SVM model for classification tasks. The input to SVC is hyperparameters gamma and C. The low value of

gamma (0.01) supplied to it can make sure, the model doesn't get overfitted. With the regularization hyperparameter setting of high value $C=100$, model will be given high penalty for misclassification in a single trained model. Thus, these settings were used to create a model.

The code is attached in [3] Appendix section and in GitHub.

Model Fit

Once the model is created with SVC, the classifier must be fitted with the data. The training data contains the scaled input training word images array and the corresponding encoded output (input/output pairs to be used in Supervised Learning approach) using fit () method in the classifier.

Model Prediction From SVC

Now, with above step, the models are trained on the image classification task, and it is ready to do predictions if supplied with the test word images. The 1/3rd of the images that got filtered as test set are now fed to predict () method of classifier. The output resulting array would be the predicted values of the SVC classifier. The code with this method call is there in [3] Appendix section.

Sample Output

The values predicted for last 326 examples are [0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0
1
0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
1 0 1 0 1 0 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1
1 1 0 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 1
0 0 1 0 0 0 1 1 1 0 0 0]

On analyzing, the output is the encoded result of 0 or 1 instead of 'Mani' or 'Priya' because we encoded the result in our training set and thus it reflects the same in output. For readable output, the encoded results can be decoded to 'Mani' or 'Priya'.

Handwriting Recognition

SVC Prediction Accuracy Analysis

From the predicted values, there are some obvious errors. So, the accuracy must be measured on this model. The scikit learn package has `accuracy_score()` method. It takes truth values and predicted values as input and returns the accuracy score as proportion to total test data.

SVC Accuracy output

Accuracy of handwriting task using SVM: 0.6779141104294478

From above, the accuracy of the given SVC classifier for this task is 67%.

SVC Classification Report

The precision and recall scores are checked using scikit learn metrics package `classification_report()` method with predicted and truth values of input.

SVC Classification Output

The classification report using SVM for Handwriting task is below

	<i>precision</i>	<i>recall</i>	<i>f1-score</i>	<i>support</i>
<i>0</i>	<i>0.69</i>	<i>0.87</i>	<i>0.77</i>	<i>203</i>
<i>1</i>	<i>0.62</i>	<i>0.37</i>	<i>0.46</i>	<i>123</i>
<i>accuracy</i>			<i>0.68</i>	<i>326</i>
<i>macro avg</i>	<i>0.66</i>	<i>0.62</i>	<i>0.62</i>	<i>326</i>
<i>weighted avg</i>	<i>0.67</i>	<i>0.68</i>	<i>0.65</i>	<i>326</i>

From above, the recall score of 37% for the second person's handwriting indicates, the model performs poorly for positive instances. Although overall accuracy is good, the model seems confused about the classification task with second handwriting.

Conclusion On SVC

The model only classifies with 67% accuracy which is poor given the handwriting recognition is a sensitive thing. So, the better classification mechanisms to be used for this task. The most

Handwriting Recognition

obvious choice will be using Convolutional Neural Networks, and the accuracy is checked with it. The model fit with CNN for the same task is explained below.

Convolutional Neural Networks

Background

The Convolutional Neural Networks shortly CNN, is a type of neural network commonly used in machine learning, particularly for tasks involving image classification, Natural language Processing and Computer Vision. So, for handwriting recognition task, this is a natural fit. For this task, the Tensorflow keras library in Python was used. Tensorflow keras library from Google exposes lot of utility methods to use CNN with a provision to use activation function (like Relu or sigmoid), maxpooling or average pool operations in hidden layers, flatten the hidden layers and softmax option for outer layer.

Image Pre-processing

The input word images that are 2-D are first converted to a grayscale image of fixed size width and height (here 28*28). Then each image array (RGB values) is divided by 255 for normalizing it. Then, the 2-D image array converted to 3-D array to add channel. These channels are useful for CNN to learn specific features or patterns from the input gray scale image. The results of these are stored in a Numpy array. So, the resultant shape would be 28*28*1. The code is present with comments in [3] Appendix section.

Model

The CNN Feed Forward sequential model is used here which performs well for image classification task like handwriting recognition. So, the input will flow in forward direction without feedback loop. Relu is used as activation function for non-linearity. This is used so that the model doesn't overfit based on the training data giving incorrect results for sparse networks.

Handwriting Recognition

Each input image array will be split up with multiple windows or patterns (kernels) of fixed size.

Code is explained in [3] Appendix section.

Sample code for creation of model is below.

```
# The activation function used here is Relu  
# The CNN below has 32 kernels of size 3x3. Also, input shape is grayscale size of 28*28 with 1  
# channel.  
# Max-pooling reduces the spatial dimensions of input.  
# Flatten flattens the output of the previous layer into a one-dimensional array.  
model = tf.keras.Sequential([  
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),  
    tf.keras.layers.MaxPooling2D((2, 2)),  
    tf.keras.layers.Flatten(),  
    tf.keras.layers.Dense(128, activation='relu'), # Hidden layer with 128 units  
    tf.keras.layers.Dense(2) # Output layer with 2 units (one for each class, handwriting task is  
for 2 classes)  
)
```

Model compilation and Fit.

The model must be compiled first. For compilation, the optimizer functions that effectively optimizes the weights based on gradient descent algorithm is required. 'Adam' is a popular optimizer for adjusting the parameters based on gradient of loss function and it was used. The loss function used was SparseCategoricalCrossentropy because the output classes are integers (0 or 1 encoded for 'Mani' or 'Priya'). Then, the model gets fitted with the training and validation data sets. Here, the epoch must be specified. Epochs are the number of times the model will be trained. The from_logits give the numerical stability without manual computation of softmax probabilities for the classification task.

Sample code is given below.

Handwriting Recognition

```
model.compile(optimizer='adam',  
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
              metrics=['accuracy'])
```

Train the model

```
model.fit(x_train, y_train_encoded, epochs=10, validation_data=(x_test, y_test_encoded))
```

Evaluation, Accuracy and Loss Analysis

The trained model above can be evaluated using `evaluate ()` method to check the performance in each epoch of this model. It returns loss and the accuracy for each epoch.

Code for Model fit

```
# Evaluate the model on the test set  
  
test_loss, test_acc = model.evaluate(x_test, y_test_encoded, verbose=2)  
  
print("\nHandwriting task test accuracy:", test_acc)
```

Output for Model fit

```
Epoch 1/10  
44/44 [=====] - 1s 15ms/step - loss: 0.6714 - accuracy: 0.6232 - val_loss: 0.6231  
- val_accuracy: 0.5925  
Epoch 2/10  
44/44 [=====] - 1s 13ms/step - loss: 0.5114 - accuracy: 0.7783 - val_loss: 0.3943  
- val_accuracy: 0.8699  
Epoch 3/10  
44/44 [=====] - 1s 14ms/step - loss: 0.3390 - accuracy: 0.8797 - val_loss: 0.3507  
- val_accuracy: 0.8035  
Epoch 4/10  
44/44 [=====] - 1s 15ms/step - loss: 0.2534 - accuracy: 0.9123 - val_loss: 0.2763  
- val_accuracy: 0.8555  
Epoch 5/10  
44/44 [=====] - 1s 15ms/step - loss: 0.1829 - accuracy: 0.9428 - val_loss: 0.1788  
- val_accuracy: 0.9335  
Epoch 6/10  
44/44 [=====] - 1s 16ms/step - loss: 0.1592 - accuracy: 0.9457 - val_loss: 0.1382  
- val_accuracy: 0.9595  
Epoch 7/10  
44/44 [=====] - 1s 18ms/step - loss: 0.1327 - accuracy: 0.9529 - val_loss: 0.1298  
- val_accuracy: 0.9566  
Epoch 8/10  
44/44 [=====] - 1s 17ms/step - loss: 0.1059 - accuracy: 0.9609 - val_loss: 0.1617  
- val_accuracy: 0.9306  
Epoch 9/10  
44/44 [=====] - 1s 17ms/step - loss: 0.1192 - accuracy: 0.9529 - val_loss: 0.1276  
- val_accuracy: 0.9538
```

Handwriting Recognition

Epoch 10/10

*44/44 [=====] - 1s 18ms/step - loss: 0.0816 - accuracy: 0.9754 - val_loss: 0.1094
- val_accuracy: 0.9538
11/11 - 0s - loss: 0.1094 - accuracy: 0.9538 - 80ms/epoch - 7ms/step*

Handriting task test accuracy: 0.9537572264671326

Prediction on Random word image

The predict () method on model can be used to predict the handwriting of random word image files it is trained on.

Sample Runs using predict () method

```
classes = ['Mani', 'Priya']
```

```
new_image_path = '/Users/manikanr/Documents/AI-Items/Handwriting-  
task/word_images/Mani_446.png'
```

```
new_image = load_image(new_image_path)
```

```
prediction = model.predict(np.array([new_image]))
```

```
predicted_label = np.argmax(prediction)
```

```
print("\nPredicted Handwriting belongs to:', classes[predicted_label])
```

```
1/1 [=====] - 0s 54ms/step
```

Predicted Handwriting belongs to: Mani

```
new_image_path = '/Users/manikanr/Documents/AI-Items/Handwriting-  
task/word_images/Priya_386.png'
```

```
new_image = load_image(new_image_path)
```

```
prediction = model.predict(np.array([new_image]))
```

```
predicted_label = np.argmax(prediction)
```

```
print("\nPredicted Handwriting belongs to:', classes[predicted_label])
```

```
1/1 [=====] - 0s 20ms/step
```

Predicted Handwriting belongs to: Priya

Conclusion on CNN classifier for handwriting task

With the handwriting task accuracy of 96% with 10 epochs, the results are impressive. So, we can conclude that traditional methods like SVM doesn't give much accuracy on models trained

Handwriting Recognition

for image classification task. Convolutional Neural Networks can give better results. With transformers and attention mechanisms, this can be further improved for such tasks. But care should be taken when feeding the handwriting of unknown person. The model will try to fit based on its trainings of 2 individuals. When it gets tested with unknown individual's handwriting, it will give results that are closer to the new one. This stresses the importance of training the model with all classes that are getting classified in Supervised Learning.

References

Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach (4th ed.). Pearson Education (US). <https://online.vitalsource.com/books/9780134671932>

OpenAI. (2022). ChatGPT [GPT-3.5]. OpenAI. <https://openai.com/chatgpt>

Appendix 1

https://github.com/manikandan18/Handwriting_Recognition

Pre-processing Step For Images. The below code converts the image files of sentences to individual word image files.

In [17]:

```
from PIL import Image, ImageDraw
import pytesseract
from pytesseract import Output
import numpy as np
import os
import cv2
import pytesseract

train_image_list = []
test_image_list = []
y_train_list = []
y_truth_list = []

# The below method would crop the individual words with the whole png file and create a new image file (.png)
# file for each word. pytesseract python library was used to do this activity. Please note, the task is to identify
# whose handwriting the image belongs to rather than identifying correct word within the image.
# The output .png file produced with each word would be like Mani_1.png, Mani_2.png etc. Here, label 'Mani' is used as
# as output of prediction once the model gets fitted.

def generate_text_images(image_path, j, image_name):
    # Load the image
    image = Image.open(image_path)

    # Perform OCR and get data including bounding boxes
    data = pytesseract.image_to_data(image, output_type=Output.DICT)
    # Create a drawing object
    draw = ImageDraw.Draw(image)
    output_directory = '/Users/manikanr/Documents/AI-Items/Handwriting-task/word_images/'
    os.makedirs(output_directory, exist_ok=True)
    i=0
    # Process the data
    for i in range(len(data['text'])):
        # Extract word bounding boxes
        word, x, y, w, h = data['text'][i], data['left'][i], data['top'][i], data['width'][i], data['height'][i]

        # Filter out non-word elements
        if word.strip() != '' and int(data['conf'][i]) > 0:
            # Crop and save the word as a separate image
            word_image = image.crop((x, y, x + w, y + h))
            word_image.save(f"{output_directory}{image_name}_{j}.png")
            if (j<=700):
                train_image = np.array(word_image)
                train_image_list.append(train_image)
                y_train_list.append(image_name)
            else:
                test_image = np.array(word_image)
                test_image_list.append(test_image)
                y_truth_list.append(image_name)
            j = j+1;
    return j

# The below code will feed the images to the method
j=1
for i in range(1,10):
    image_path = "/Users/manikanr/Documents/AI-Items/Handwriting-task/Handwritten_Mani_"+str(i)+".png"
    j = generate_text_images(image_path, j, 'Mani')

j = 1
```

```
for k in range(1,9):
    image_path = "/Users/manikanr/Documents/AI-Items/Handwriting-task/Handwritten_Priya_"+s
    tr(k)+".png"
    j = generate_text_images(image_path, j, 'Priya')

print("\nAll handwritten sentences in multiple images are transformed successfully into individual word images.")
```

All handwritten sentences in multiple images are transformed successfully into individual word images.

Check and Plot the individual word images

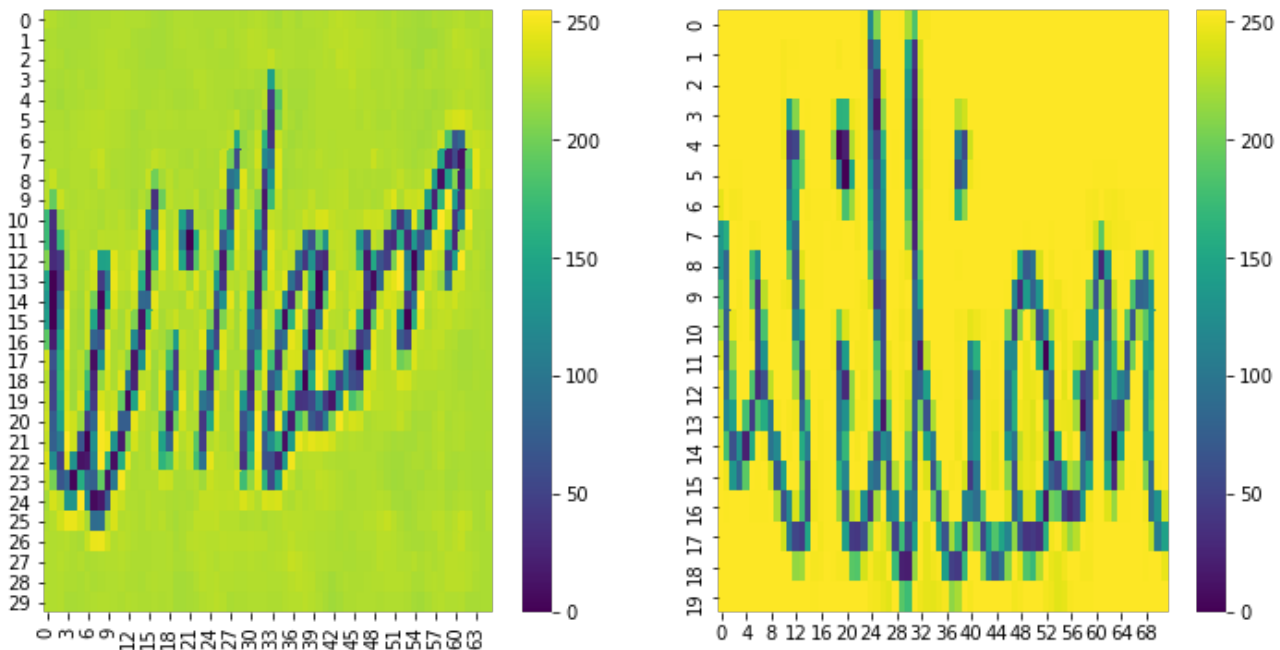
In [18]:

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

print(len(train_image_list))
print(len(test_image_list))
plt.figure(figsize=(12, 6))
plt.subplot(1,2,1)
image = Image.open('/Users/manikanr/Documents/AI-Items/Handwriting-task/word_images/Priya_6.png')
image_array = np.array(image)
print(image_array.shape)
channel = image_array[:, :, 0]
sns.heatmap(channel, cmap='viridis')

plt.subplot(1,2,2)
image = Image.open('/Users/manikanr/Documents/AI-Items/Handwriting-task/word_images/Mani_1.png')
image_array = np.array(image)
print(image_array.shape)
channel = image_array[:, :, 0]
sns.heatmap(channel, cmap='viridis')
plt.show()
```

```
1400
326
(30, 66, 4)
(20, 72, 4)
```



Word Image Arrays are flattened from 3-D to 1-D array with re-shape and shapes are checked in below Python Code

In [21]:

```
# Convert the list of images to a numpy array
train_image_array = np.array(train_image_list, dtype=object)
```



```
test_image_array = np.array(test_image_list, dtype=object)
```

In [22]:

```
# Assuming image_array is your list of 3D image arrays
train_flattened_images = []
test_flattened_images = []

for img in train_image_array:
    # Flatten each 3D image array into a 1D array
    flattened_img = img.reshape(-1)
    # Append the flattened image to the list
    train_flattened_images.append(flattened_img)

# Convert the list of flattened images to a NumPy array
train_flattened_images_array = np.array(train_flattened_images, dtype=object)
print("\nShape of train word image array:", train_flattened_images_array.shape)

for img in test_image_array:
    # Flatten each 3D image array into a 1D array
    flattened_img = img.reshape(-1)
    # Append the flattened image to the list
    test_flattened_images.append(flattened_img)

# Convert the list of flattened images to a NumPy array
test_flattened_images_array = np.array(test_flattened_images, dtype=object)
print("\nShape of test word image array:", test_flattened_images_array.shape)
print("\n")
```

Shape of train word image array: (1400,)

Shape of test word image array: (326,)

The below code is used to for below changes 1. Padding input word image array 2. Encoding output variable 3. SVM Model Creation 4. SVM Model Fit 5. Using SVM classifier to predict output for 326 word images in test set

In [24]:

```
from sklearn import svm
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler

# Create a MinMaxScaler object
scaler = MinMaxScaler(feature_range=(0, 1))

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the string labels to numerical labels
y_encoded = label_encoder.fit_transform(y_train_list)

clf = svm.SVC(gamma=0.01, C=100)

# Find the maximum shape among all elements in train_flattened_images_array
max_train_shape = max(sample.shape for sample in train_flattened_images_array)
max_test_shape = max(sample.shape for sample in test_flattened_images_array)
max_shape = max(max_train_shape, max_test_shape)

# Create a new list to store padded samples
padded_train_samples = []
padded_test_samples = []

# Pad each sample to match the maximum shape
for sample in train_flattened_images_array:
    pad_width = [(0, max_dim - sample_dim) for max_dim, sample_dim in zip(max_shape, sample.shape)]
    padded_sample = np.pad(sample, pad_width, mode='constant', constant_values=0)
    padded_train_samples.append(padded_sample)

# Convert the list of padded samples to a NumPy array
```

```

padded_array = np.array(padded_train_samples)

# Reshape the padded array to 2 dimensions
num_samples = padded_array.shape[0]
num_features = np.prod(max_shape)
padded_array_reshaped = padded_array.reshape(num_samples, num_features)

x = padded_array_reshaped

# Fit and transform the data
x_scaled = scaler.fit_transform(x)
y_encoded = np.array(y_encoded)
clf.fit(x_scaled, y_encoded)

padded_test_samples = []

# Find the maximum shape among all elements in train_flattened_images_array
# max_shape = max(sample.shape for sample in test_flattened_images_array)

# Pad each sample to match the maximum shape
for sample in test_flattened_images_array:
    pad_width = [(0, max_dim - sample_dim) for max_dim, sample_dim in zip(max_shape, sample.shape)]
    padded_sample = np.pad(sample, pad_width, mode='constant', constant_values=0)
    padded_test_samples.append(padded_sample)

# Convert the list of padded samples to a NumPy array
padded_array = np.array(padded_test_samples)

# Reshape the padded array to 2 dimensions
num_samples = padded_array.shape[0]
num_features = np.prod(max_shape)
padded_array_reshaped = padded_array.reshape(num_samples, num_features)

test_x = padded_array_reshaped

test_x_scaled = scaler.fit_transform(test_x)

predict_last_326 = clf.predict(test_x_scaled)
print("The values predicted for last 326 examples are {}".format(predict_last_326))

```

```

The values predicted for last 326 examples are [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0
0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
1 1 0 0 1 1 1 0 0 0 0 1 1 0 1 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 1 1 0 0
1 0 1 0 1 1 1 0 0 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0 1 1 0 1 0 0]

```

Accuracy, precision and Recall for SVM Classifier is checked in below 2 code snippets

In [26]:

```

from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt

y_truth_encoded = label_encoder.fit_transform(y_truth_list)
# Calculate accuracy
accuracy = accuracy_score(y_truth_encoded, predict_last_326)
print("\nAccuracy of handwriting task using SVM:", accuracy)
print("\n")

```

Accuracy of handwriting task using SVM: 0.6779141104294478

In [10]:

```

from sklearn.metrics import classification_report

```

```
# Generate a classification report for svm
report = classification_report(y_truth_encoded, predict_last_326)

# Print the classification report
print("\nThe classification report using SVM for Handwriting task is below\n")
print(report)
```

The classification report using SVM for Handwriting task is below

	precision	recall	f1-score	support
0	0.69	0.87	0.77	203
1	0.62	0.37	0.46	123
accuracy			0.68	326
macro avg	0.66	0.62	0.62	326
weighted avg	0.67	0.68	0.65	326

Handwriting recognition using Convolutional Neural Networks The below code converts all 1726 word images to fixed Grayscale image (of size 28*28) and normalize it by dividing it by 255. It adds the channel dimension as well. Also, it splits the word images with training set of 80% and test set of 20%

In [11]:

```
# Handwriting recognition using Neural Networks

from sklearn.model_selection import train_test_split

# Function to extract the label from the file name
def get_label(file_name):
    return file_name.split('_')[0] # Extract the label before the underscore

# Load and preprocess the image data
def load_image(file_path, target_size=(28, 28)):
    image = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
    image = cv2.resize(image, target_size)
    image = image.astype(np.float32) / 255.0 # Normalize to [0, 1]
    image = np.expand_dims(image, axis=-1) # Add a channel dimension
    return image

# List all image files in the directory
image_dir = '/Users/manikanr/Documents/AI-Items/Handwriting-task/word_images/'
image_files = [os.path.join(image_dir, file) for file in os.listdir(image_dir) if file.endswith('.png')]

# Extract labels from the file names
labels = [get_label(file.split('/')[-1]) for file in image_files]

# Split the data into training and testing sets
x_train_files, x_test_files, y_train, y_test = train_test_split(image_files, labels, test_size=0.2, random_state=42)

# Load and preprocess the training and testing images
x_train = np.array([load_image(file) for file in x_train_files])
x_test = np.array([load_image(file) for file in x_test_files])
```

The below code does the following. 1. CNN Model creation with activation function of Relu, maxpooling and flattening. The number of kernels used is 32 and size of each is 3*3. It uses final dense layer with 2 units specific to number of classes in output 'Mani' or 'Priya'. 2. Compiles the model 3. CNN Model fit with training and validation set 4. Printing the accuracy achieved

In [12]:

```
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the string labels to numerical labels
```

```

y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Define the model architecture with convolutional neural network
# The activation function used here is Relu
# The CNN below has 32 kernels of size 3x3. Also, input shape is grayscale size of 28*28
with 1 channel
# Max-pooling reduces the spatial dimensions of input.
# Flatten flattens the output of the previous layer into a one-dimensional array.

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'), # Hidden layer with 128 units
    tf.keras.layers.Dense(2) # Output layer with 2 units (one for each class, handwriti
ng task is for 2 classes)
])

# Compile the model
# 1. 'adam' is a type of stochastic gradient descent (SGD) optimization algorithm that ef
ficiently
# updates the parameters based on the gradient of loss function
# 2. The SparseCategoricalCrossentropy is the loss function used when the output classes
are integers. Here 0 and
# 1 for 'Mani' and 'Priya' respectively.

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train_encoded, epochs=10, validation_data=(x_test, y_test_encoded))

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test_encoded, verbose=2)
print('\nHandriting task test accuracy:', test_acc)

```

```

Epoch 1/10
44/44 [=====] - 1s 15ms/step - loss: 0.6714 - accuracy: 0.6232 -
val_loss: 0.6231 - val_accuracy: 0.5925
Epoch 2/10
44/44 [=====] - 1s 13ms/step - loss: 0.5114 - accuracy: 0.7783 -
val_loss: 0.3943 - val_accuracy: 0.8699
Epoch 3/10
44/44 [=====] - 1s 14ms/step - loss: 0.3390 - accuracy: 0.8797 -
val_loss: 0.3507 - val_accuracy: 0.8035
Epoch 4/10
44/44 [=====] - 1s 15ms/step - loss: 0.2534 - accuracy: 0.9123 -
val_loss: 0.2763 - val_accuracy: 0.8555
Epoch 5/10
44/44 [=====] - 1s 15ms/step - loss: 0.1829 - accuracy: 0.9428 -
val_loss: 0.1788 - val_accuracy: 0.9335
Epoch 6/10
44/44 [=====] - 1s 16ms/step - loss: 0.1592 - accuracy: 0.9457 -
val_loss: 0.1382 - val_accuracy: 0.9595
Epoch 7/10
44/44 [=====] - 1s 18ms/step - loss: 0.1327 - accuracy: 0.9529 -
val_loss: 0.1298 - val_accuracy: 0.9566
Epoch 8/10
44/44 [=====] - 1s 17ms/step - loss: 0.1059 - accuracy: 0.9609 -
val_loss: 0.1617 - val_accuracy: 0.9306
Epoch 9/10
44/44 [=====] - 1s 17ms/step - loss: 0.1192 - accuracy: 0.9529 -
val_loss: 0.1276 - val_accuracy: 0.9538
Epoch 10/10
44/44 [=====] - 1s 18ms/step - loss: 0.0816 - accuracy: 0.9754 -
val_loss: 0.1094 - val_accuracy: 0.9538
11/11 - 0s - loss: 0.1094 - accuracy: 0.9538 - 80ms/epoch - 7ms/step

Handriting task test accuracy: 0.9537572264671326

```

In []:

The below code shows `is` used to show prediction results of CNN Classifier using random image files

In [13]:

```
classes = ['Mani', 'Priya']
new_image_path = '/Users/manikanr/Documents/AI-Items/Handwriting-task/word_images/Mani_446.png'
new_image = load_image(new_image_path)
prediction = model.predict(np.array([new_image]))
predicted_label = np.argmax(prediction)
print('\nPredicted Handwriting belongs to:', classes[predicted_label])
```

1/1 [=====] - 0s 71ms/step

Predicted Handwriting belongs to: Mani

In [16]:

```
new_image_path = '/Users/manikanr/Documents/AI-Items/Handwriting-task/word_images/Priya_386.png'
new_image = load_image(new_image_path)
prediction = model.predict(np.array([new_image]))
predicted_label = np.argmax(prediction)
print('\nPredicted Handwriting belongs to:', classes[predicted_label])
```

1/1 [=====] - 0s 17ms/step

Predicted Handwriting belongs to: Priya

In []: