Signature Verification

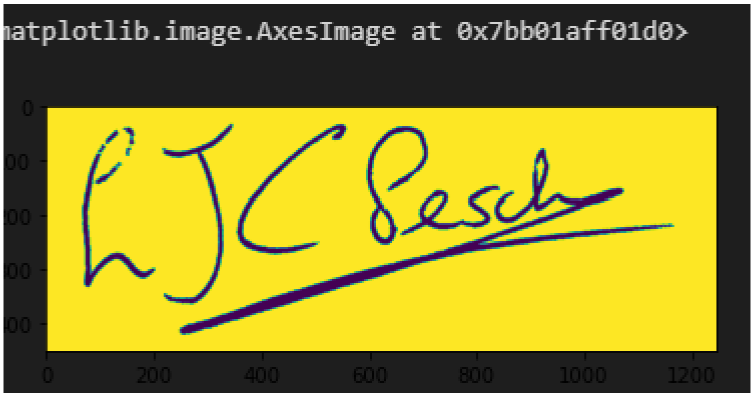
Steps:
1.Data Collection
2.Data Preprocessing
3.Model Train
4.Evaluate the model
5.Deploy the model

1.Data Collection

The data is collected from the kaggle dataset , which contains real and forged folders that can be shown the original and fake signature.

2. Data Preprocessing

Using the matplotlib library the image has been shown in the below format and also import all the dataset through the glob library.



```
atplotlib.image.AxesImage at 0x7bb01aff01d0>
```

3.Model Training

Before training the model the data is split into training and testing data, 70% data is used for training and 30% of data is used for the testing purpose.

1. Importing Libraries

- `from keras.models import Sequential`: Imports the Sequential model class from Keras, which allows you to create a linear stack of layers.
- `from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout`: Imports various layer types from Keras, including Convolutional, Pooling, Flatten, Dense, and Dropout layers.

- `from keras_preprocessing.image import ImageDataGenerator`: Imports the ImageDataGenerator class from Keras, which allows for easy data augmentation and preprocessing of image data.
- `from sklearn.metrics import confusion_matrix as CM`: Imports the confusion_matrix function from scikit-learn and assigns it an alias of `CM`.
- `from keras.optimizers import Adam`: Imports the Adam optimizer from Keras, which is a popular optimization algorithm.

2. Creating the Neural Network Model

- Model Definition: The model is defined as a sequential stack of layers (`Sequential()`).
- Layers:
  - Convolutional Layers:
    - `Conv2D(64, (3,3), input_shape=(224,224,3), activation='relu')`: Adds a convolutional layer with 64 filters, each of size 3x3, with ReLU activation function. The `input_shape` parameter specifies the shape of input images.
    - `MaxPooling2D(3,3)`: Adds a max pooling layer with pool size 3x3.
    - `Conv2D(32, (3,3), activation='relu')`: Adds another convolutional layer with 32 filters, each of size 3x3, with ReLU activation function.
    - `MaxPooling2D(2,2)`: Adds another max pooling layer with pool size 2x2.
  - Flatten Layer: `Flatten()`: Flattens the input, transforming the pooled feature maps into a 1D array.
  - Dense Layers:
    - `Dense(128, activation='relu')`: Adds a fully connected (dense) layer with 128 units and ReLU activation.
    - `Dropout(rate=0.3)`: Adds a dropout layer with a dropout rate of 0.3, which helps prevent overfitting by randomly dropping a fraction of input units.
    - `Dense(2, activation='softmax')`: Adds the output layer with 2 units (assuming binary classification) and softmax activation, which outputs probability scores for each class.

3. Compiling the Model

- Compilation: `network.compile(optimizer=Adam(lr=0.001), loss='binary_crossentropy', metrics=['accuracy'])`: Compiles the model, specifying the optimizer, loss function, and evaluation metric.

- Optimizer: Uses the Adam optimizer with a learning rate of 0.001.
- Loss Function: Specifies binary cross entropy as the loss function, suitable for binary classification tasks.
- Metrics: Specifies accuracy as the evaluation metric.

4. Summary

- Model Summary: `network.summary()`: Prints a summary of the model architecture, including the type and shape of each layer, as well as the total number of parameters.

4.Evaluate the model

To evaluate the model to check the accuracy score and give conclusion to the model.

5.Deployment of the Model

Model can be deploy in various methods like web application, streamlit application and upload the image and check whether the image id forged or real signature.