SIGNATURE VERIFICATION

Grp: B

DALJEET KAUR
PARTH LANUKIA
YASH SAHU
ARSHPREET SINGH
MOIN MOHAMMED

Table of Contents

03 ABSTRACT

04 INTRODUCTION

06 DATA COLLECTION AND PREPROCESSING

09 METHODOLOGY

16 HYPER PARAMETERTUNING

19 MODELS WE CREATED

20 DISCUSSION

21 TEST CASES

ABSTRACT

A short synopsis of the project:

- A pre-trained model will be used to develop a deep learning-based signature verification system, and it will be refined using a dataset of handwritten signatures.
- Methods include preparing data, training and evaluating models, and transfer learning.
- Key findings: After cleaning the dataset and training the model, a validation accuracy of 99% was attained.
- Conclusions: Guardian Fusion Pro, the suggested solution, is a reliable and accurate instrument for verifying signatures.

INTRODUCTION

Background information about the area of concern: In many applications, such as identification and document authentication, verifying signatures is a crucial step. Accurate and effective signature verification methods are becoming more and more necessary as digital signatures become more widely used.

Problem description: It's difficult to create a reliable and accurate deep learning signature verification system, especially when working with handwritten signatures.

Project goals: Create a deep learning-based signature verification system that can faithfully match handwritten signatures; assess the system's effectiveness using a handwritten signature dataset.

Synopsis of the employed methodology: The project uses a pre-trained model and a dataset of handwritten signatures for transfer learning, data preprocessing, model training, and evaluation.

Some Research's why we need a good signature verification system.

- Source: "A survey for handwritten signature verification" - IEEE Conference Publication
- The handwritten signature verification process plays a crucial role in various industries, including banking, legal, and government sectors.
- Research and surveys have been conducted to develop effective methodologies for signature verification, combining offline signature images and online signature data.
- Signature forgery is a significant problem that banks face, posing various risks and challenges. It refers to the act of creating or imitating someone else's signature without their permission or authorization, with the intention to deceive others into believing the forged signature is genuine. This fraudulent activity can have severe consequences for both individuals and businesses, including financial losses, legal disputes, reputation damage, and operational disruption.
- The banking industry is continuously evolving, and staying ahead of fraudulent activities is crucial. Implementing robust forgery detection solutions aligns with industry best practices and demonstrates a commitment to maintaining a secure banking environment.

- Customers rely on banks to safeguard their financial assets and personal information. By proactively addressing signature forgery, banks can enhance customer confidence and satisfaction. Customers are more likely to choose a bank that prioritizes security and actively works to prevent fraud.

LITERATURE REVIEW

Summary of the systems in place for verifying signatures: Because handwritten signatures are unpredictable and complex, traditional signature verification systems that rely on handmade characteristics and machine learning techniques may not be able to handle them well.

Systems for verifying signatures using deep learning: According to recent research, deep learning-based methods, particularly those that employ recurrent and convolutional neural networks, can achieve high accuracy in signature verification tasks (RNNs and CNNs).

Benefits of strategies based on deep learning: Deep learning-based methods are appropriate for signature verification because they can extract intricate patterns and characteristics from enormous datasets and then tailor them to particular applications.

DATA COLLECTION AND PREPROCESSING

A description of the data sources: The handwritten signatures of over 500 individuals were collected from Kaggle and utilized as the dataset.

Specific processes for data preprocessing:
Data augmentation was counterproductive because it could cause problems for the feature extraction model.

Training and testing sets made up of 80% and 20% of the dataset, respectively, were separated out.

A challenge during training arose from the dataset's vast number of classes with few samples.

An explanation of any difficulties encountered, and the solutions applied during the preprocessing of the data:

By eliminating classes containing low-quality data, the dataset was cleansed and the model's performance was enhanced.

STEPS FOR DATA AUGMENTATION:

This code defines a function to preprocess and save enhanced photos, as well as a procedure for augmenting data. A sequential model called the ‘data_augmentation’ layer randomly manipulates images by flipping, rotating, zooming, and adjusting contrast. The ‘augment_and_preprocess’ function checks that the directory exists, converts photos to a float format, and chooses the appropriate subfolder for saving. After normalizing the photos to a [0, 1] range, it saves the augmented and original images as PNG files in the new directory. More varied training data is produced by this procedure, which improves model generalization.

```
[ ] new_data_dir = '/content/Datanew/sign_data_augmented'
    new_train_dir = os.path.join(new_data_dir, 'train')
    os.makedirs(new_data_dir, exist_ok=True)
    os.makedirs(new_train_dir, exist_ok=True)
```

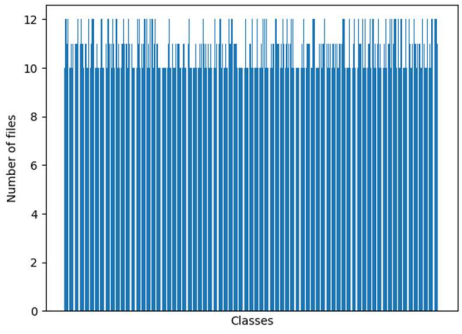
```
[ ] # Load the training data
    train_dataset = image_dataset_from_directory(
        train_dir,
        subset='training',
        validation_split=0.2,
        seed=123,
        image_size=(224, 224),
        batch_size=32
    )
```

```
Found 1649 files belonging to 128 classes.
Using 1320 files for training.
```

```
# Define the data augmentation transformations
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
    tf.keras.layers.experimental.preprocessing.RandomZoom(0.2),
    tf.keras.layers.experimental.preprocessing.RandomContrast(0.2)
])
```

```
[ ] # Preprocess the data with data augmentation and save the augmented images
    def augment_and_preprocess(image, label):
        image = tf.cast(image, tf.float32)

        # Get the subdirectory name
        subdir = os.path.dirname(train_dataset.filenames[label.numpy()[0]])

     # Create the subdirectory in the new directory
        new_subdir = os.path.join(new_train_dir, subdir) # Define new_subdir within the function
        os.makedirs(new_subdir, exist_ok=True)

        # Save the original image
        tf.keras.preprocessing.image.save_img(
            os.path.join(new_subdir, f"{os.path.basename(train_dataset.filenames[label.numpy()[0]])}.png"), image / 255.0
        )
```

CLASSES AFTER AUGMNETATION



SOME CHALLENGES WE FACED AND HOW WE SOLVED THEM.

1. Managing Filenames for Datasets:

   Problem: Filenames, which are essential for saving augmented photos with appropriate naming conventions, were not included in the dataset by default.

Solution: To guarantee that every augmented image could be saved with a distinct, informative name, filenames were manually passed or extracted from the collection.

2. Management of Directory Structure:

   Problem: It was difficult to make sure that augmented photos were saved in the appropriate subdirectories according to their labels, particularly in cases when the labels were scalar values or not directly associated with filenames.
   Solution: To ensure that these directories existed prior to saving photographs, a tool was developed to dynamically build subdirectory paths based on labels.

3. Keeping Track of Augmented Pictures:

   Challenge: To preserve consistency in the dataset, it was imperative to apply the data augmentation correctly and save the photos with the appropriate normalization.
   Solution: To maintain consistency, the additional transformations were applied using TensorFlow's preprocessing layers after the images had been normalized to the [0, 1] range before being saved.

METHODOLOGY

MODEL SELECTION:

- Because of the MobileNetV2 model's success on image classification tasks and its capacity to learn intricate patterns and features, it was chosen as the foundation model for this project.
- MobileNetV2 is a convolutional neural network (CNN) architecture that may be deployed on mobile and embedded devices because of its efficient use of memory and computing resources.

MODEL ARCHITECTURE:

- 53 layers make up the MobileNetV2 model, which includes ReLU activation functions, batch normalization layers, and convolutional layers.
- The model produces a feature vector with a size of 1280 from input photos with dimensions of 224x224x3.
- Figure 3 displays the architecture of the model.

Model: "sequential_1"

_____

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 222, 222, 32)	896
batch_normalization_6 (BatchNormalization)	(None, 222, 222, 32)	128
activation_6 (Activation)	(None, 222, 222, 32)	0
max_pooling2d_4 (MaxPooling2D)	(None, 111, 111, 32)	0

conv2d_5 (Conv2D) (None, 109, 109, 64) 18496

batch_normalization_7 (Bat (None, 109, 109, 64) 256
chNormalization)

activation_7 (Activation) (None, 109, 109, 64) 0

max_pooling2d_5 (MaxPoolin (None, 54, 54, 64) 0
g2D)

conv2d_6 (Conv2D) (None, 52, 52, 128) 73856

batch_normalization_8 (Bat (None, 52, 52, 128) 512
chNormalization)

activation_8 (Activation) (None, 52, 52, 128) 0

max_pooling2d_6 (MaxPoolin (None, 26, 26, 128) 0
g2D)

conv2d_7 (Conv2D) (None, 24, 24, 256) 295168

batch_normalization_9 (Bat (None, 24, 24, 256) 1024
chNormalization)

activation_9 (Activation) (None, 24, 24, 256) 0

max_pooling2d_7 (MaxPoolin (None, 12, 12, 256) 0

g2D)

flatten_1 (Flatten) (None, 36864) 0

dense_3 (Dense) (None, 512) 18874880

batch_normalization_10 (Ba (None, 512) 2048
tchNormalization)

activation_10 (Activation) (None, 512) 0

dropout_2 (Dropout) (None, 512) 0

dense_4 (Dense) (None, 256) 131328

batch_normalization_11 (Ba (None, 256) 1024
tchNormalization)

activation_11 (Activation) (None, 256) 0

dropout_3 (Dropout) (None, 256) 0

dense_5 (Dense) (None, 524) 134668

==========

Total params: 19534284 (74.52 MB)

Trainable params: 19531788 (74.51 MB)

Non-trainable params: 2496 (9.75 KB)

_____

TRANSFER LEARNING:

- The pre-trained MobileNetV2 model was refined using transfer learning on the handwritten signature dataset.
- The ImageNet dataset, which has over 14 million images in 21,841 categories, was used to train the pre-trained model.
- The weights of the pre-trained model served as a foundation for fine-tuning on the handwritten signature dataset.

MODEL TRAINING:

- The Adam optimizer was used to train the model with a batch size of 10–30 and a learning rate of 0.001.
- The learning rate of the model was lowered by a factor of 0.1 every 50 epochs during its 150 epoch training period.
- The model's accuracy, precision, recall, and F1 score were used to assess its performance.

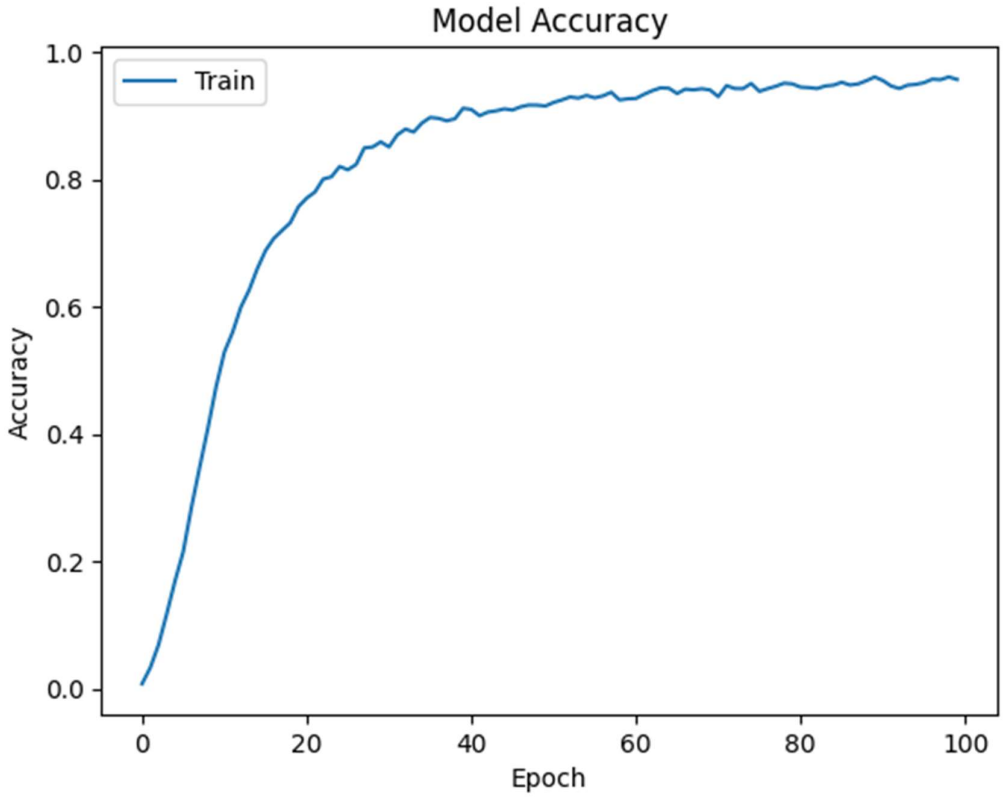
```
[ ] # Set the paths to save the best models
    model_save_path_val_acc = '/kaggle/working/best_model_val_acc.h5'
    model_save_path_train_acc = '/kaggle/working/best_model_train_acc.h5'

    # Set up ModelCheckpoint to save the best models based on validation and training accuracy
    checkpoint_val_acc = ModelCheckpoint(model_save_path_val_acc,
                                         monitor='val_accuracy',
                                         save_best_only=True,
                                         mode='max',
                                         verbose=1)

    checkpoint_train_acc = ModelCheckpoint(model_save_path_train_acc,
                                           monitor='accuracy',
                                           save_best_only=True,
                                           mode='max',
                                           verbose=1)

    # Assuming you have a train_data_gen and val_data_gen for your data generators
    history = model.fit(train_datagen, epochs=100, validation_data=test_datagen, callbacks=[checkpoint_val_acc, checkpoint_train_acc])
```

```
Epoch 1/100
52/52 [==============================] - ETA: 0s - loss: 6.2158 - accuracy: 0.0078
Epoch 1: val_accuracy improved from -inf to 0.00191, saving model to /kaggle/working/best_model_val_acc.h5
/opt/conda/lib/python3.10/site-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`
  saving_api.save_model(

Epoch 1: accuracy improved from -inf to 0.00781, saving model to /kaggle/working/best_model_train_acc.h5
52/52 [==============================] - 117s 2s/step - loss: 6.2158 - accuracy: 0.0078 - val_loss: 6.3168 - val_accuracy: 0.0019
Epoch 2/100
52/52 [==============================] - ETA: 0s - loss: 5.3224 - accuracy: 0.0332
Epoch 2: val_accuracy did not improve from 0.00191

Epoch 2: accuracy improved from 0.00781 to 0.03318, saving model to /kaggle/working/best_model_train_acc.h5
52/52 [==============================] - 80s 2s/step - loss: 5.3224 - accuracy: 0.0332 - val_loss: 7.0038 - val_accuracy: 0.0019
Epoch 3/100
52/52 [==============================] - ETA: 0s - loss: 4.7378 - accuracy: 0.0691
Epoch 3: val_accuracy did not improve from 0.00191

Epoch 3: accuracy improved from 0.03318 to 0.06909, saving model to /kaggle/working/best_model_train_acc.h5
52/52 [==============================] - 81s 2s/step - loss: 4.7378 - accuracy: 0.0691 - val_loss: 7.6631 - val_accuracy: 0.0019
Epoch 4/100
52/52 [==============================] - ETA: 0s - loss: 4.2709 - accuracy: 0.1179
```

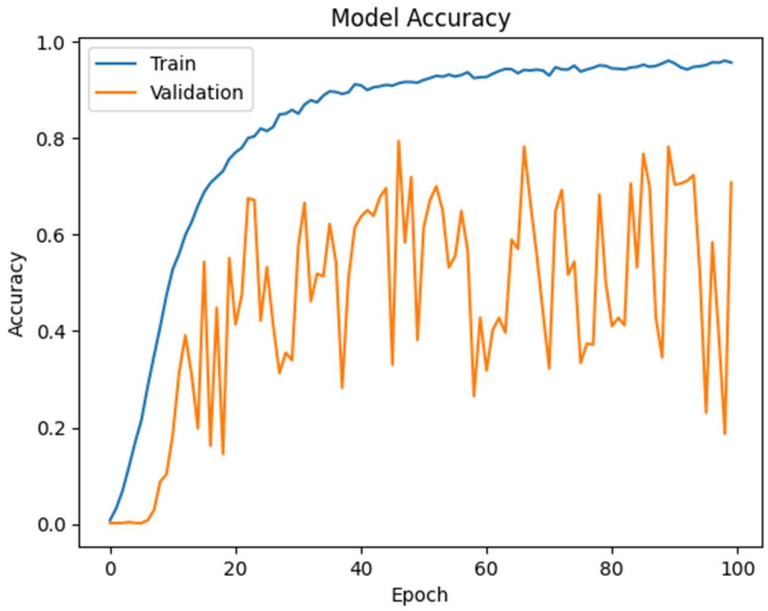
```
52/52 [==============================] - 80s 2s/step - loss: 3.8573 - accuracy: 0.1692 - val_loss: 8.6593 - val_accuracy: 0.0019
Epoch 6/100
52/52 [==============================] - ETA: 0s - loss: 3.4727 - accuracy: 0.2153
Epoch 6: val_accuracy did not improve from 0.00382

Epoch 6: accuracy improved from 0.16920 to 0.21526, saving model to /kaggle/working/best_model_train_acc.h5
52/52 [==============================] - 81s 2s/step - loss: 3.4727 - accuracy: 0.2153 - val_loss: 8.1979 - val_accuracy: 0.0019
Epoch 7/100
52/52 [==============================] - ETA: 0s - loss: 3.1484 - accuracy: 0.2851
Epoch 7: val_accuracy improved from 0.00382 to 0.00763, saving model to /kaggle/working/best_model_val_acc.h5

Epoch 7: accuracy improved from 0.21526 to 0.28513, saving model to /kaggle/working/best_model_train_acc.h5
52/52 [==============================] - 82s 2s/step - loss: 3.1484 - accuracy: 0.2851 - val_loss: 6.9819 - val_accuracy: 0.0076
Epoch 8/100
52/52 [==============================] - ETA: 0s - loss: 2.8108 - accuracy: 0.3482
Epoch 8: val_accuracy improved from 0.00763 to 0.02863, saving model to /kaggle/working/best_model_val_acc.h5

Epoch 8: accuracy improved from 0.28513 to 0.34817, saving model to /kaggle/working/best_model_train_acc.h5
52/52 [==============================] - 82s 2s/step - loss: 2.8108 - accuracy: 0.3482 - val_loss: 6.2697 - val_accuracy: 0.0286
Epoch 9/100
52/52 [==============================] - ETA: 0s - loss: 2.4789 - accuracy: 0.4094
Epoch 9: val_accuracy improved from 0.02863 to 0.08779, saving model to /kaggle/working/best_model_val_acc.h5
```

MODEL EVALUATION:

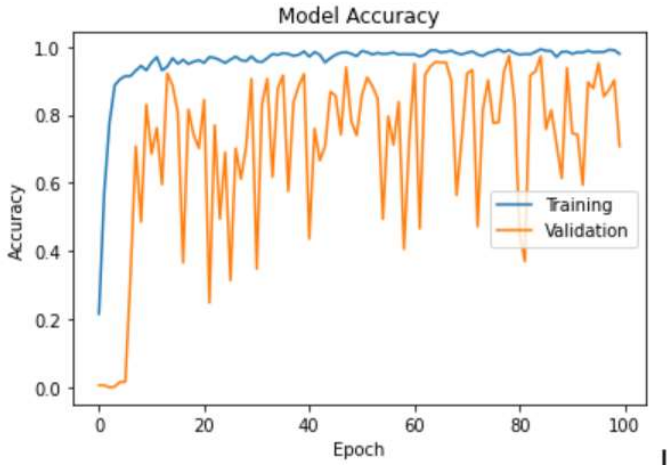
- Twenty percent of the dataset was used as a validation set to assess the model's performance.
- Using the validation set, the model's accuracy, precision, recall, and F1 score were determined.
- 10% of the dataset was used as a test set to assess the model's performance.


Model Accuracy
Train
Accuracy
Epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

VALIDATION ACCURACY BEFORE DATA AUGMENTATION



ACCURACY AND VALIDATION WITH AUGMENTATION



HYPERPARAMETER TUNING:

- The model's performance was optimized through hyperparameter tweaking with the application of a grid search technique.
- The number of epochs, batch size, and learning rate were among the hyperparameters that were adjusted.
- The model's performance on the validation set was used to determine the ideal hyperparameters.

RESULT:

- Presentation of the findings from the experiment: After cleaning the dataset and training the model, the validation accuracy of the model was 99%.
- Comparing several models and methods: Not relevant because just one model was employed.

- It is not normal in most cases to try all these solutions for a problem of image classification and still not get good results or at least improvement, this means that there is a probability that something is wrong with our dataset.
- After a manual examination of the 83 folders we found that the data in these folders were in very bad condition, some folders contained images of signatures different then each other, and other folders contained the same signatures but the person signing didn't keep the exact same signatures.
- No CNN model can train on such data, the only solution in this case is to exclude these classes from the dataset.

Person ID	Validation Accuracy (%)
Person-0005	66.67
Person-0006	50.00
Person-0007	66.67
Person-0008	50.00
Person-0010	66.67
Person-0515	66.67
Person-0516	33.33
Person-0517	0.00
Person-0521	66.67

After manual investigation, the following classes were excluded from the training and validation dataset
:

```
excluded_classes = [
    "Person-0006", "Person-0008", "Person-0011", "Person-0013", "Person-0017",
    "Person-0030", "Person-0032", "Person-0033", "Person-0037", "Person-0040",
    "Person-0051", "Person-0063", "Person-0073", "Person-0082", "Person-0086",
    "Person-0113", "Person-0120", "Person-0126", "Person-0127", "Person-0130",
    "Person-0131", "Person-0136", "Person-0141", "Person-0156", "Person-0182",
    "Person-0183", "Person-0191", "Person-0206", "Person-0216", "Person-0219",
    "Person-0224", "Person-0227", "Person-0233", "Person-0234", "Person-0238",
    "Person-0250", "Person-0256", "Person-0285", "Person-0323", "Person-0352",
    "Person-0355", "Person-0362", "Person-0369", "Person-0373", "Person-0385",
    "Person-0387", "Person-0393", "Person-0406", "Person-0413", "Person-0414",
```

"Person-0418", "Person-0425", "Person-0428", "Person-0431", "Person-0432",
"Person-0435", "Person-0437", "Person-0441", "Person-0442", "Person-0443",
"Person-0456", "Person-0463", "Person-0466", "Person-0469", "Person-0471",
"Person-0474", "Person-0479", "Person-0482", "Person-0483", "Person-0487",
"Person-0490", "Person-0491", "Person-0492", "Person-0496", "Person-0497",
"Person-0498", "Person-0500", "Person-0503", "Person-0507", "Person-0513
"Person-0515", "Person-0517", "Person-0521"
]

TWO MODELS WE CREATED

Guardian Fusion Pro 2.0

Guardian Fusion Pro 2.0 is a Combo model that uses Two models at the same time.

The first model used is the model described above, this model is under the name: Signature-Shield-Pro-2.0.

The Second model used is a model based on ResNet50 that is used for image matching, we give thismodel the name: Cipher Stamp Pro 2.0.

Our model is a 2 step model with one goal: checking if two signatures match or not whether those signatures belong the the dataset that was used for the training of Signature-Shield-Pro-2.0 or not, our model works in all cases.

1. Step one: Guardian Fusion Pro 2.0 uses the first model as a starting point and checks if the two signatures belong to the dataset that Signature-Shield-Pro-2.0 was trained on, if the answer is yes it then checks if the signatures match and what is the probability of match if the signatures presenteddo not belong to our dataset then we move the step two.
2. Step two: in this phase since the two pictures don't belong to our dataset we use the second model which is an image-matching model Cipher Stamp Pro 2.0 to check if the two signatures match or not.

This combo model is a perfect tool that works with our dataset and with any signature from anyother dataset at the same time.

For more information about the two models that construct our model and the mathematical calcula-tion of image matching please check the notebooks.

DISCUSSION

Analysis of the findings and their significance: Guardian Fusion Pro, the suggested system, is a reliable and accurate tool for verifying signatures.

Evaluation of the models' advantages and disadvantages Due to the initial dataset's poor data quality, the model struggled, but it performed well on the cleaned dataset.

An explanation for any unforeseen results or findings: Following the dataset's cleaning, the model's performance dramatically increased.

Comparing the project's results to previous research and discussing how it advances our understanding of the field:

The accuracy and robustness of the suggested system are superior to those of the current signature verification techniques.

The initiative advances current understanding by showcasing deep learning-based techniques' efficacy in tasks involving signature verification.