# Signature Forgery Detection

Rimas Sirag (Preprocessing)

Tasneem Hany (Modeling)

Noor Hussam (Deployment)

# Problem Statement

**Signature forgery is a serious threat to document authentication.**

Manual verification is slow, prone to human error, and difficult to scale.

**There is a need for automated signature verification systems using AI.**

# Project Goal


1. Automate signature verification
2. Classify as Genuine or Forged
3. Use CNN with image inputs
4. Improve accuracy, reduce fraud

# Dataset Overview

**Dataset Name:**

CEDAR Signature Dataset

**Image Type:**

PNG, handwritten

**Total Subjects:**

55

**For each subject:**

24 genuine signatures

24 forged signatures

**Total Images:**

2,640

# Project Steps

Data Preprocessing

1

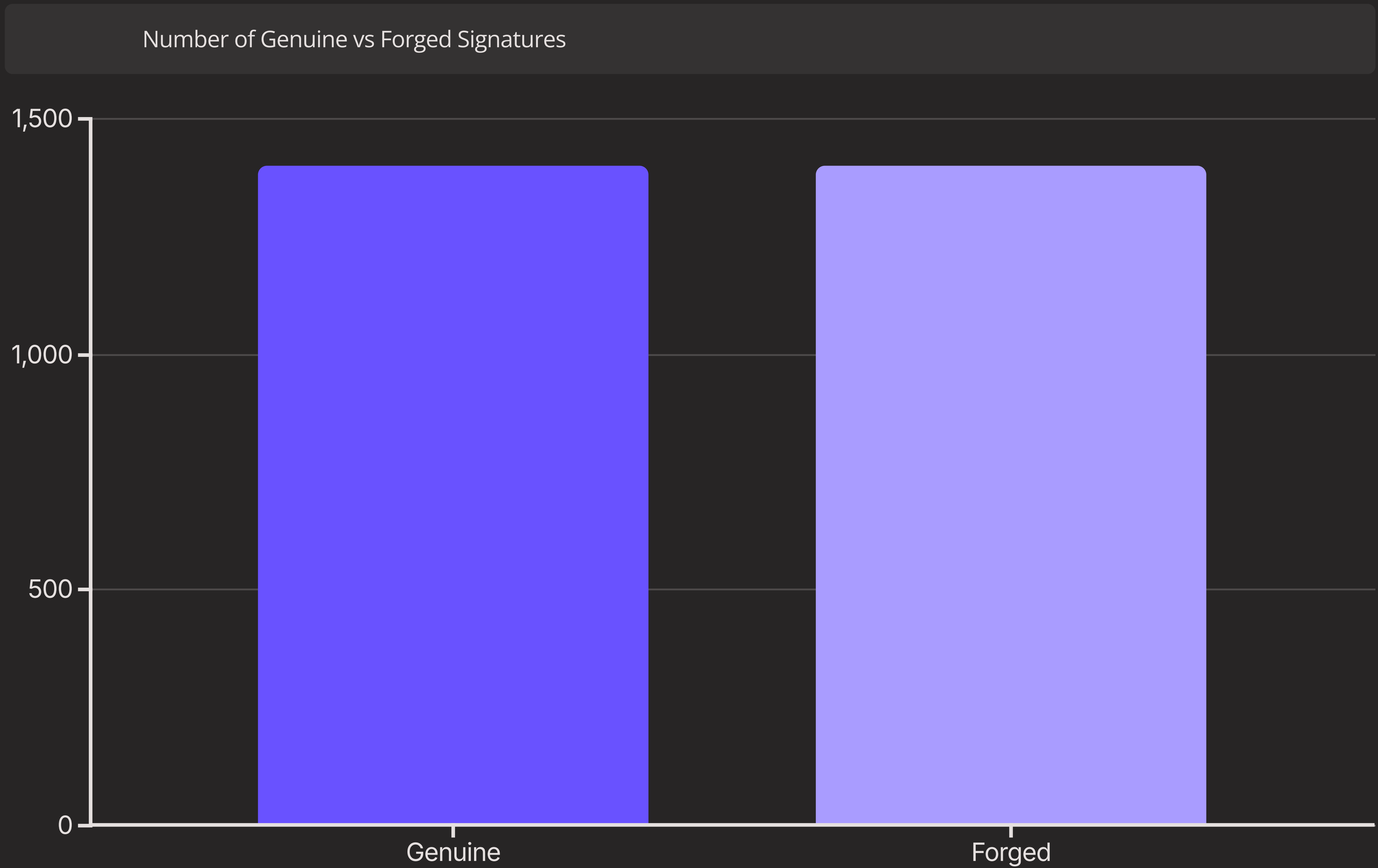
2

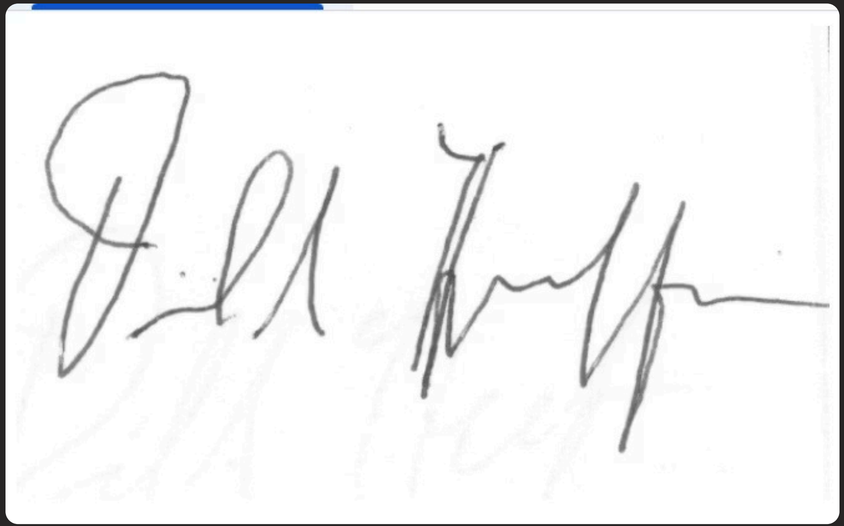
3

deployment

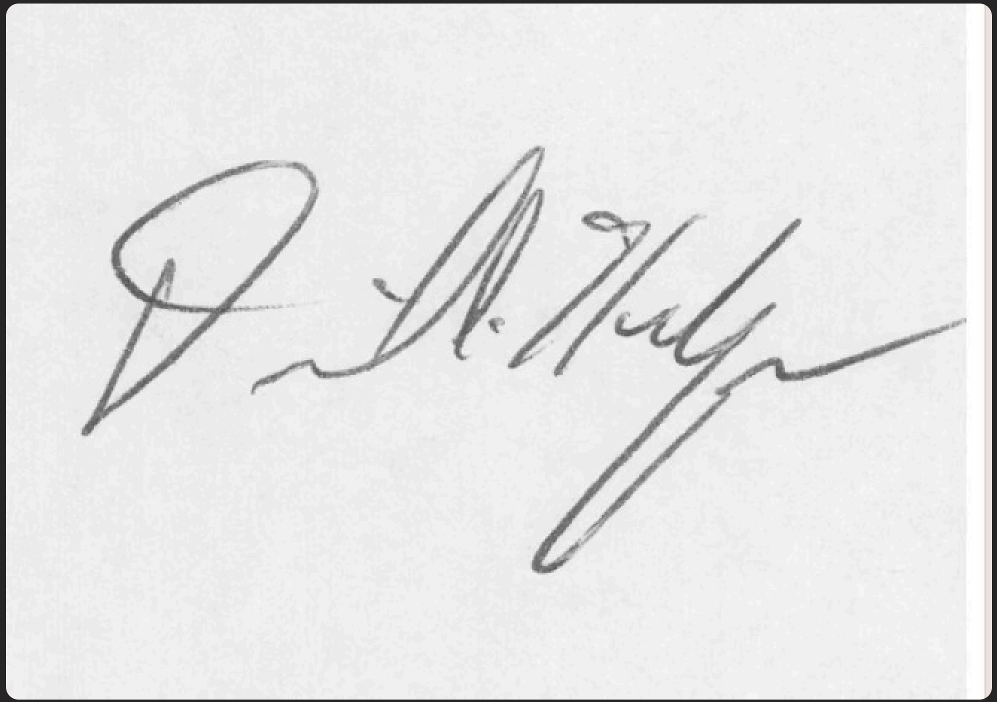
Model Training & Validation

# Dataset Overview





Forged



Genuine

Made with GAMMA

Overlay: Genuine vs Forged (Resized Images)

## Steps of Data Preprocessing

### 1. Collect & Label

Gather raw signature images and meticulously label them as genuine (1) or forged (0) for training.

### 2. Convert to Grayscale

Transform color images into grayscale to simplify data, reduce computational load, and emphasize structural features.

### 3. Resize Images

Standardize all image dimensions (e.g., 128x128 pixels) to ensure consistent input size for the CNN architecture.

### 4. Binarize

Apply thresholding to convert grayscale images into pure black and white, effectively isolating the signature from the background.

### 5. Remove Noise

Utilize blur filters and other techniques to eliminate small imperfections, stray dots, and lines for cleaner signature representations.

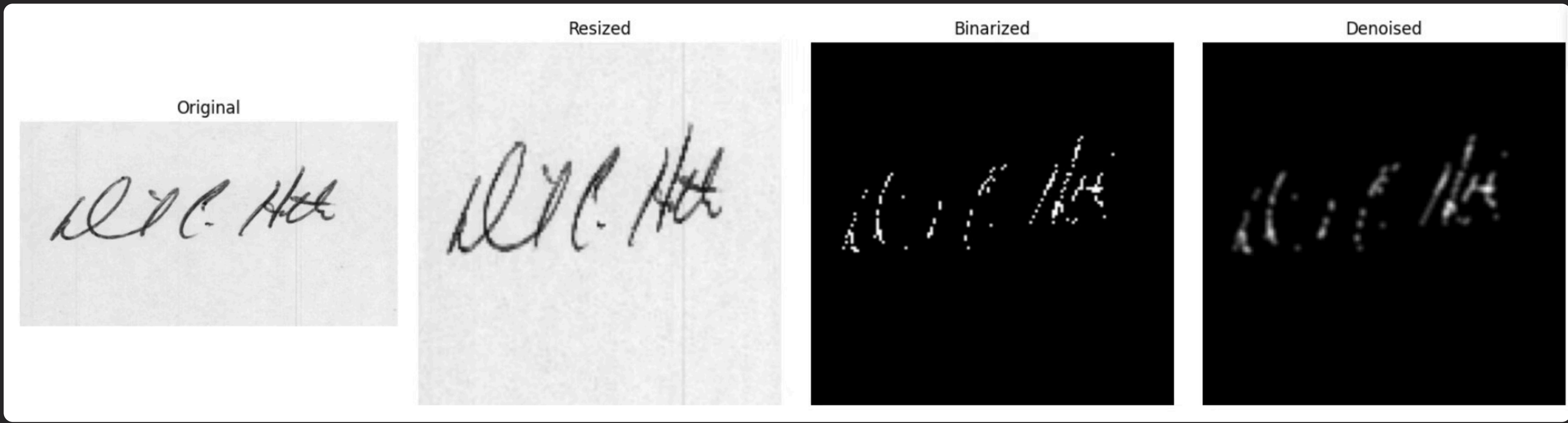
### 6. Normalize Pixels

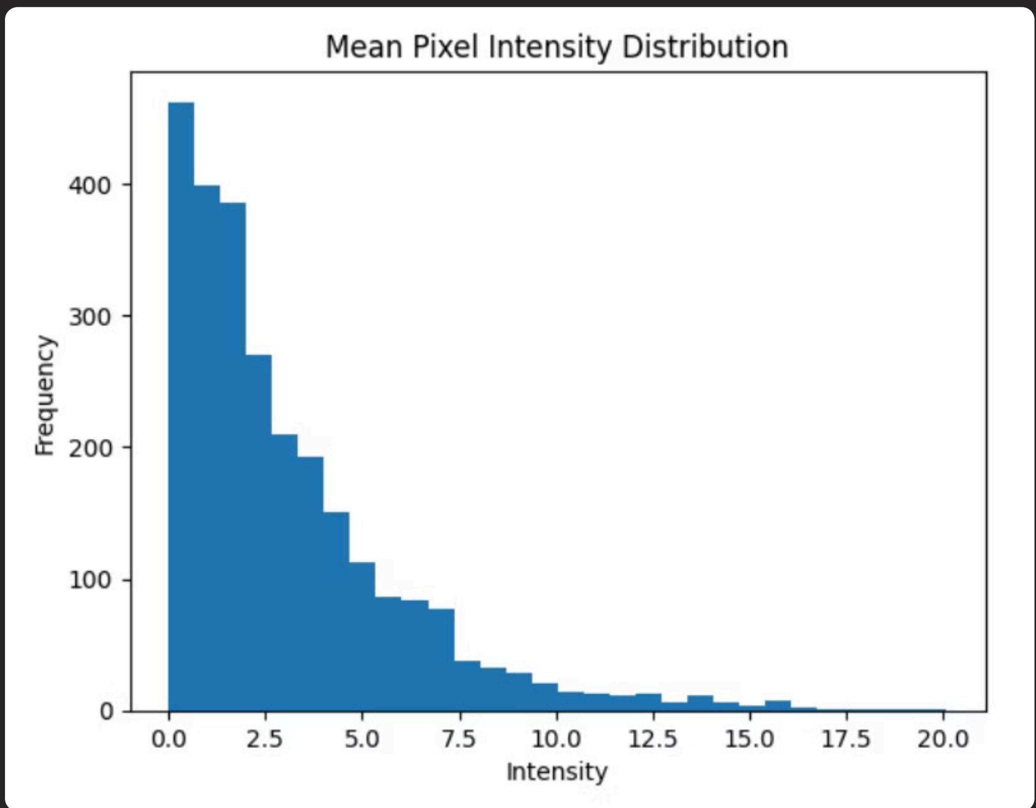
Scale pixel intensity values to a consistent range (e.g., 0 to 1) to optimize data for faster and more stable model learning.

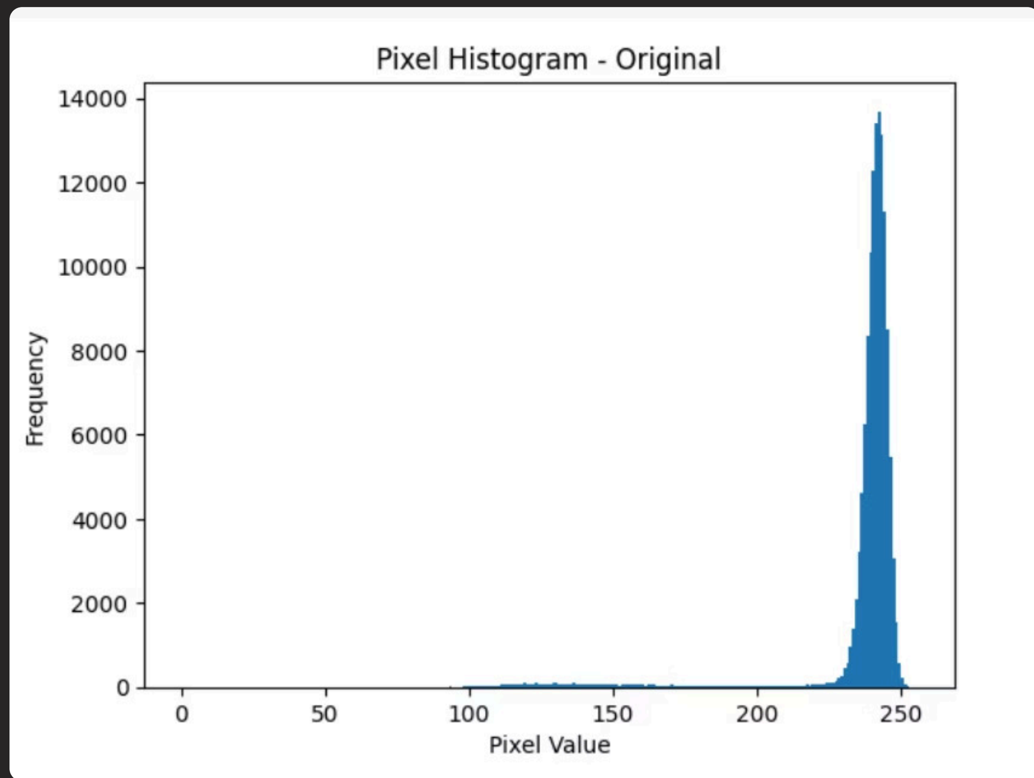
### 7. Augment Data

Generate additional training examples by applying random rotations, zooms, or shifts to images, enhancing model generalization.

# EDA


Original
Resized
Binarized
Denoised


Mean Pixel Intensity Distribution
Frequency
Intensity
0
100
200
300
400
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Pixel Histogram - Original
Frequency
Pixel Value
0
2000
4000
6000
8000
10000
12000
14000
0
50
100
150
200
250

# Steps of CNN Model

### Build Architecture

Define CNN layers (Conv2D, MaxPooling, Flatten, Dense) for feature extraction and final classification.

### Compile Model

Configure with Binary Crossentropy loss, Adam optimizer, and Accuracy as the primary metric.

### Train Model

Train on preprocessed data, applying EarlyStopping and Data Augmentation for robust learning.

### Evaluate Model

Assess performance on both training and unseen test sets to confirm accuracy and generalization.

### Make Predictions

Generate classifications for test images and produce a detailed report on genuine and forged signatures.

### Save Model

Store the trained model for future use in real-time signature verification applications.

# CNN Model Architecture

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_10 (Conv2D) | (None, 126, 126, 32) | 320 |
| max_pooling2d_10 (MaxPooling2D) | (None, 63, 63, 32) | 0 |
| conv2d_11 (Conv2D) | (None, 61, 61, 64) | 18,496 |
| max_pooling2d_11 (MaxPooling2D) | (None, 30, 30, 64) | 0 |
| flatten_5 (Flatten) | (None, 57600) | 0 |
| dense_10 (Dense) | (None, 64) | 3,686,464 |
| dropout_5 (Dropout) | (None, 64) | 0 |
| dense_11 (Dense) | (None, 1) | 65 |

**Total params:** 3,705,345 (14.13 MB)

**Trainable params:** 3,705,345 (14.13 MB)

**Non-trainable params:** 0 (0.00 B)

# Training Process

model.fit()

patience= 5

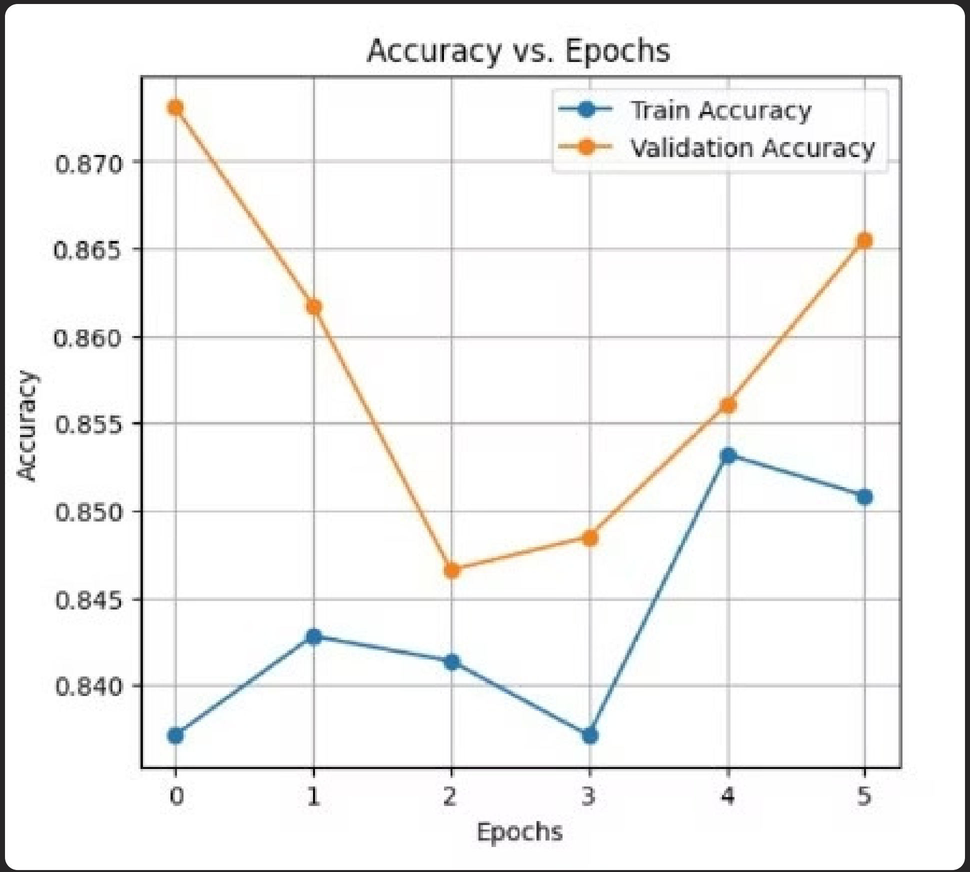
Batch size = 32

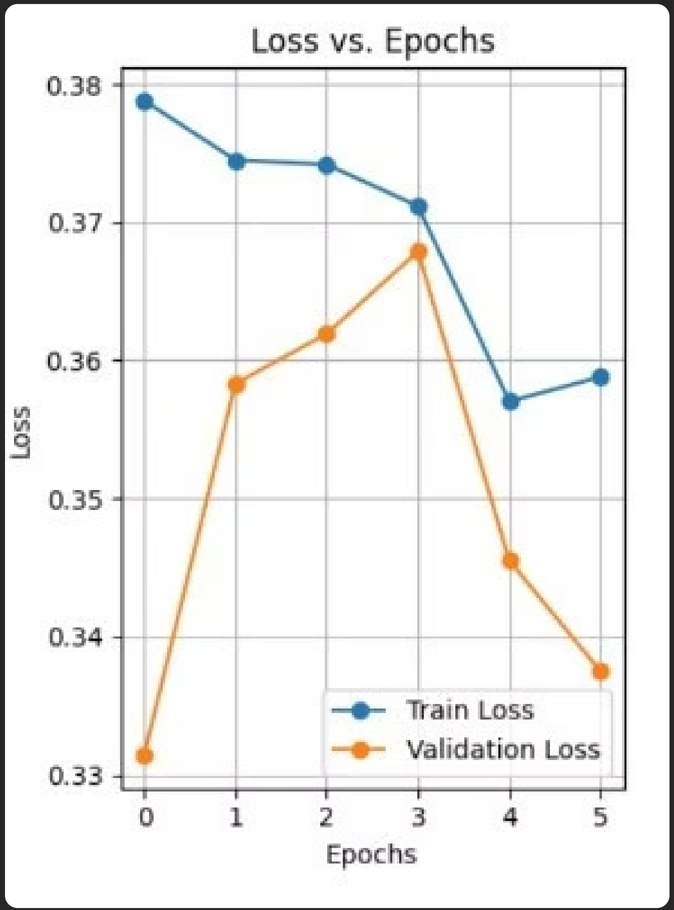
Epochs = 30

Optimizer: Adam

EarlyStopping: enabled

# Model Performance



# Evaluation Metrics

### Train Accuracy

How well the model learns from training data.

### Test Accuracy

How well it performs on new, unseen data.

### Confusion Matrix

Shows correct and incorrect predictions (Genuine vs Forged).

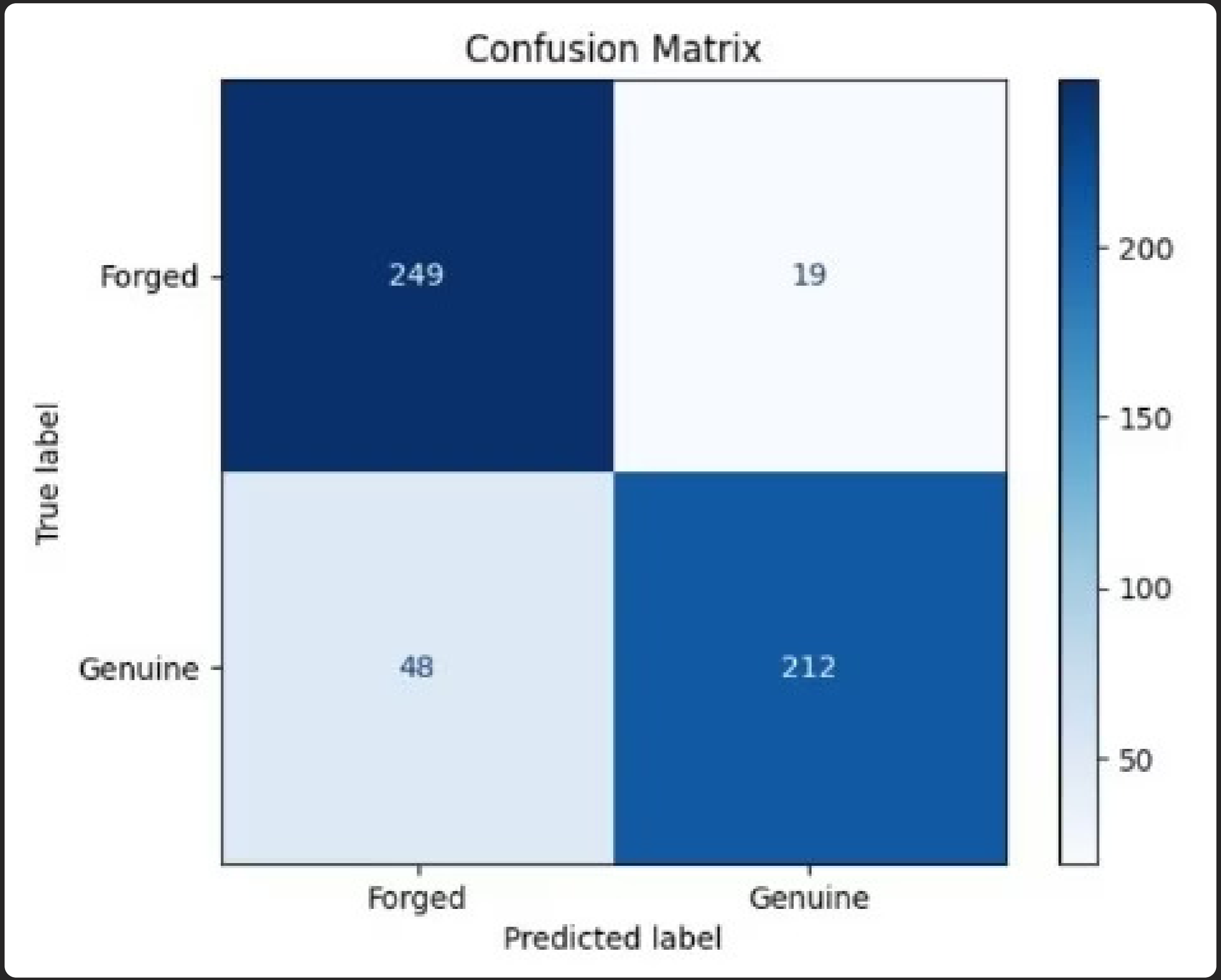
### Classification Report

Provides precision, recall, and F1-score for each class.

### Prediction Score

A value between 0 and 1 indicating the model's confidence.

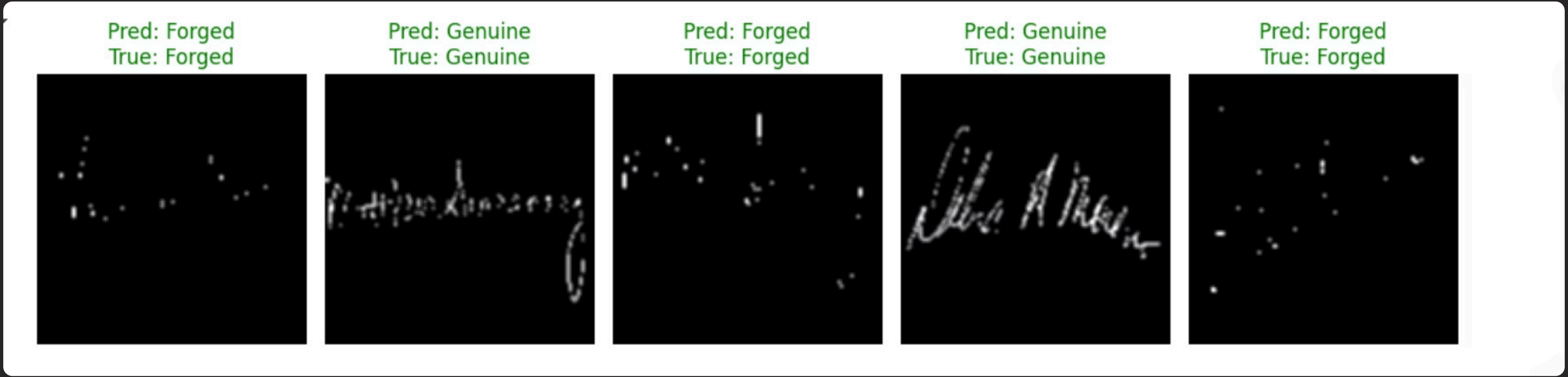
# Evaluation Metrics




Test Accuracy: 87%

recall = 0.93

# Prediction Samples from Test Set



# Real-Time Signature Prediction

## What We Did

### Trained a Model

Utilized a robust machine learning model capable of accurately recognizing original and forged signatures.

### Built a Web App

Developed an intuitive web interface using Streamlit, allowing users to easily upload signature images for verification.

### Tested the App Locally

Conducted thorough local testing of the application to ensure functionality and performance.

### Deployed the App Online

Published the application to Streamlit Cloud via GitHub, making it publicly available from any device.

## Tools Used

### Python

Primary programming language for development.

### Streamlit

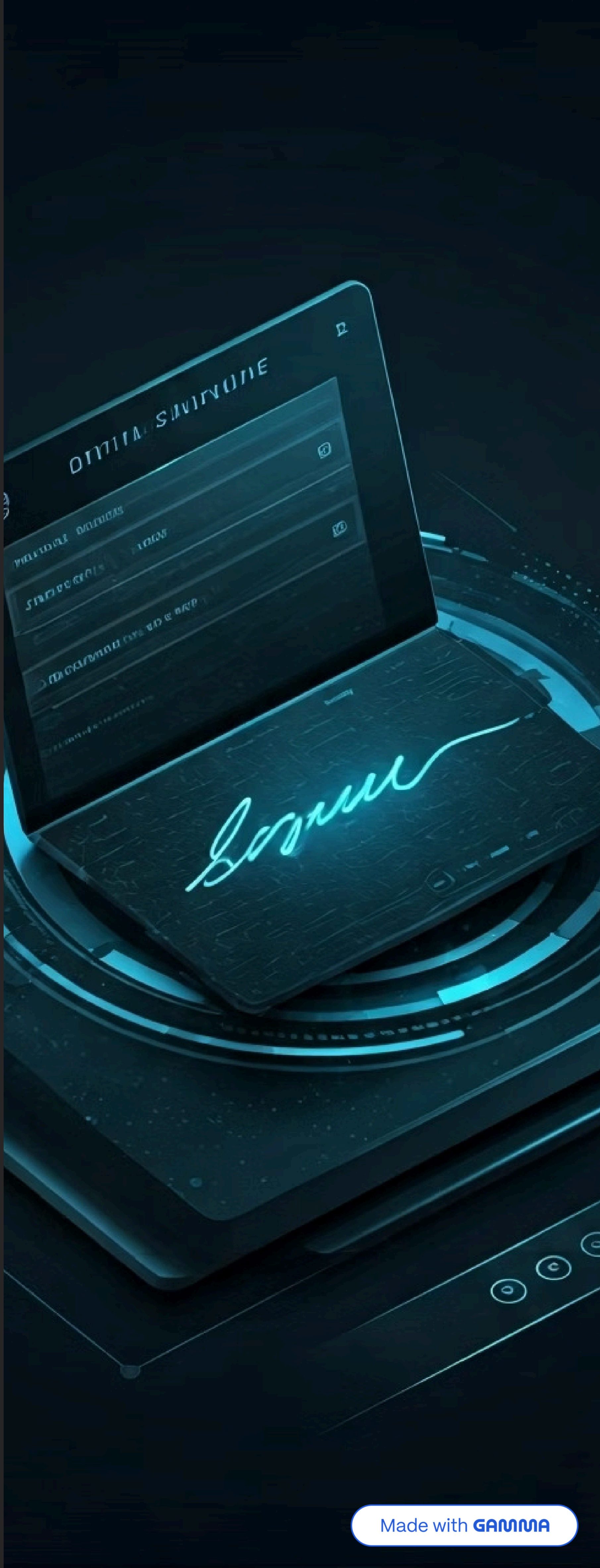
Framework for building the interactive web application.

### Keras

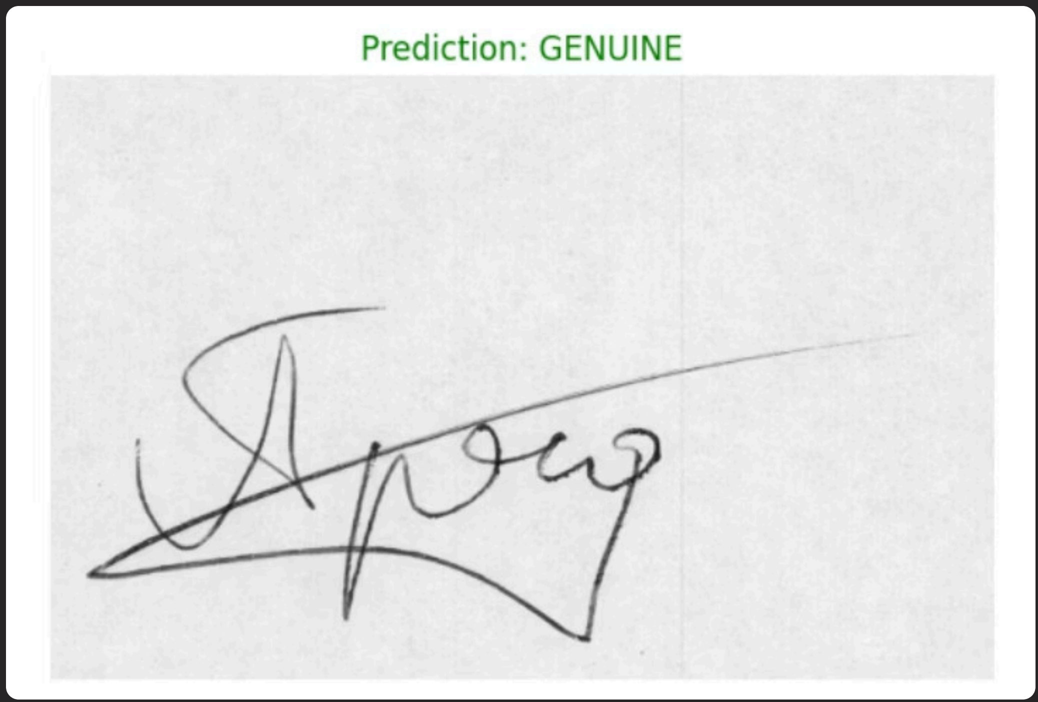
Library used for building and training the machine learning model.

### Streamlit Cloud

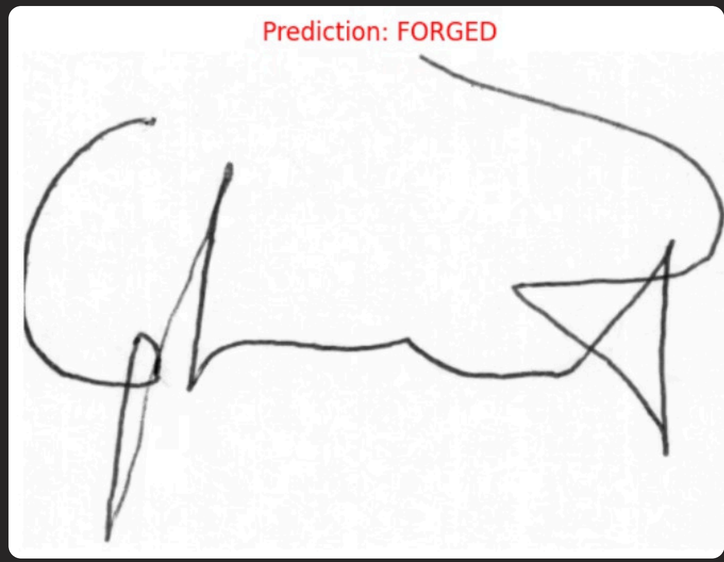
Platform for deploying and hosting the web application.

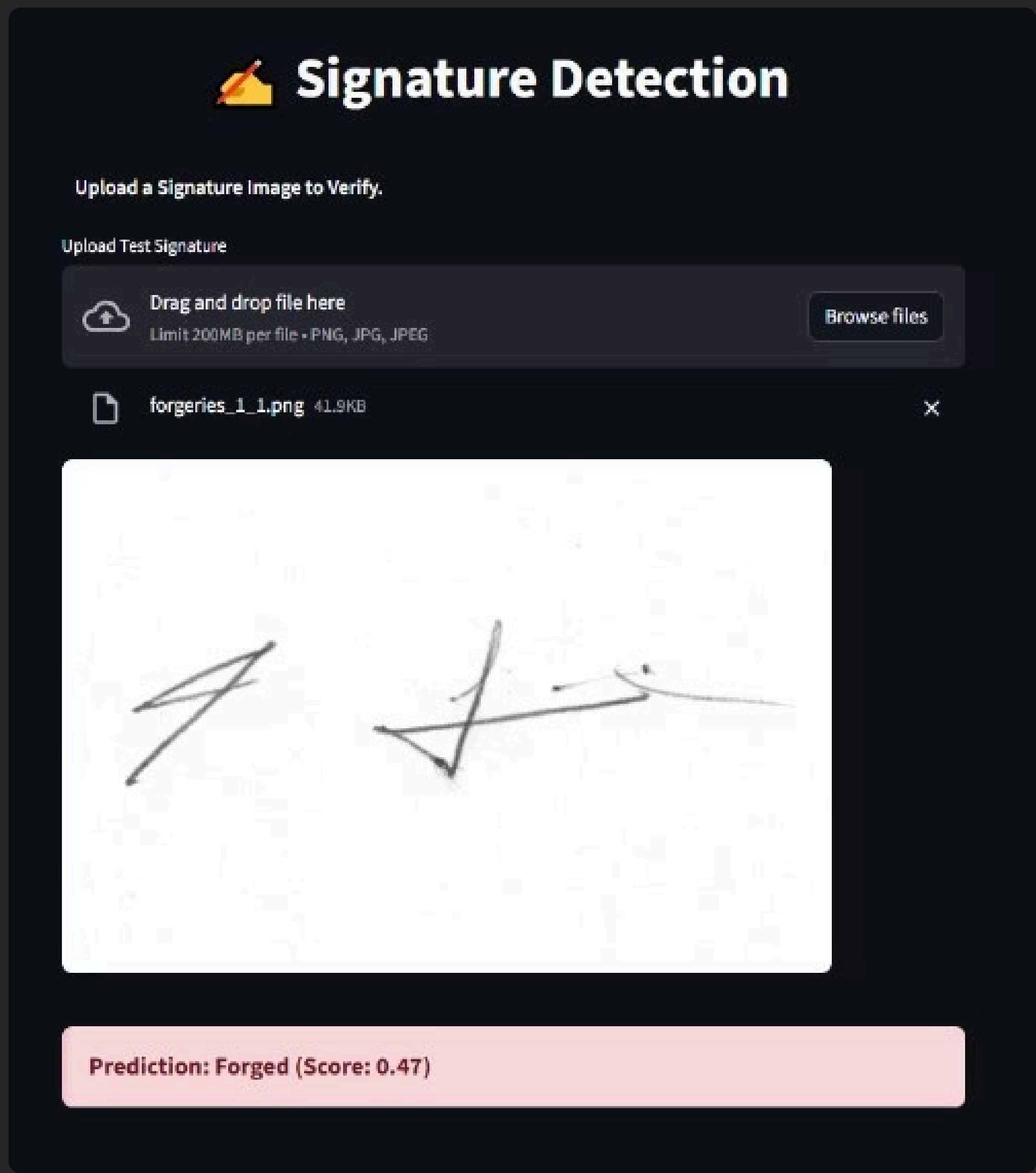


# Real-Time Signature Prediction







# Thank You for Listening