Peer Evaluation System UI/UX

We worked on the signature recognition model and developed a system to detect whether a signature is real or forged.

The code for the model and the output screenshots are attached below: -

- **Python code: -**

```
import os
import cv2
import numpy as np
from google.colab import drive
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from keras.callbacks import EarlyStopping
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

drive.mount('/content/drive')



# Define image dimensions
IMG_WIDTH = 128
IMG_HEIGHT = 64

# Load images from the dataset directory
def load_images(folder):
    images = []
    labels = []

    # Iterate over files in the dataset folder
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
```

```
        img = cv2.resize(img, (IMG_WIDTH, IMG_HEIGHT))
        img = img / 255.0  # Normalize pixel values to [0,1]
        images.append(img)

        # Determine label based on filename
        if 'original' in filename:
            labels.append(0)  # Label for original signatures
        elif 'forgeries' in filename:
            labels.append(1)  # Label for forged signatures

    return np.array(images), np.array(labels)

# Load dataset
dataset_folder = '/content/drive/MyDrive/Sign_Data/Train'
X, y = load_images(dataset_folder)
X = X.reshape(-1, IMG_WIDTH, IMG_HEIGHT, 1)  # Reshape for CNN input


# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# CNN model
model = Sequential()

# Convolutional layer 1
model.add(Conv2D(32, (3, 3), input_shape=(IMG_WIDTH, IMG_HEIGHT, 1),
activation='relu'))
model.add(BatchNormalization())
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))

# Convolutional layer 2
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

# Convolutional layer 3
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten layer
model.add(Flatten())

# Dense layer
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))

# Output layer
model.add(Dense(1, activation='sigmoid'))

# Compile model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Define early stopping callback
early_stopping = EarlyStopping(
    monitor='val_loss',  # Can also use 'val_accuracy'
    patience=5,  # Number of epochs with no improvement after which training
will be stopped
    restore_best_weights=True  # Restore model weights from the epoch with the
best value of the monitored quantity
)

# Fit the model
history = model.fit(
    datagen.flow(X_train, y_train, batch_size=32),
    validation_data=(X_test, y_test),
    epochs=30,
```

```
    callbacks=[early_stopping]
)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy * 100:.2f}%')

# Save the model
model.save('/content/drive/MyDrive/Sign_Data/signature_recognition_model.h5')

# Load dataset for testing
test_folder = '/content/drive/MyDrive/Sign_Data/Test'
X_test, y_test = load_images(test_folder)
X_test = X_test.reshape(-1, IMG_WIDTH, IMG_HEIGHT, 1)  # Reshape for
CNN input

# Load the trained model
model = load_model('signature_recognition_model.h5')

# Make predictions
y_pred = (model.predict(X_test) > 0.5).astype("int32")

# Evaluate the model
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

accuracy = accuracy_score(y_test, y_pred)
print(f'Test Accuracy: {accuracy * 100:.2f}%')

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=['Original', 'Forged']))

# Compute and plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Original',
'Forged'], yticklabels=['Original', 'Forged'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

```
plt.title('Confusion Matrix')
plt.show()


# Define image dimensions
IMG_WIDTH = 128
IMG_HEIGHT = 64


# Preprocess a single image
def preprocess_image(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)  # Load image in grayscale
    img = cv2.resize(img, (IMG_WIDTH, IMG_HEIGHT))  # Resize image to match training input size
    img = img / 255.0  # Normalize pixel values to [0,1]
    img = img.reshape(1, IMG_WIDTH, IMG_HEIGHT, 1)  # Reshape to add batch dimension and channels for CNN input
    return img

# Load the trained model
model = load_model('signature_recognition_model.h5')

# Path to the image you want to test
image_path = '/content/drive/MyDrive/Sign_Data/Test/forgeries_34_24.png'  # Replace with the path to your single image

# Preprocess the image
image = preprocess_image(image_path)

# Make prediction
prediction = model.predict(image)

# Determine class based on prediction
if prediction > 0.5:
    print("The signature is forged.")
else:
    print("The signature is original.")
```

- **The screenshots of the output: -**