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: -**
  1. **For signature verification -**

```
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.")
```

2. **For signature mapping -**

Model 1 -

```
import os
import cv2
import numpy as np
from google.colab import drive
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout, BatchNormalization
from tensorflow.keras.utils import to_categorical
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 = 256
IMG_HEIGHT = 256

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

    # Iterate over files in the dataset folder
    for filename in os.listdir(folder):
        if 'original' in filename:  # Only process original signatures
            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)
```

```
            # Extract person number from filename
            person_id = filename.split('_')[1]  # Assuming the filename is
original_Per_imgno

            # Assign a unique label to each person
            if person_id not in person_labels:
                person_labels[person_id] = len(person_labels)

            labels.append(person_labels[person_id])

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


# Load dataset
dataset_folder = '/content/drive/MyDrive/Sign_Data/Real'  # Update with your
dataset folder
X, y = load_images(dataset_folder)

# Reshape images for CNN input
X = X.reshape(-1, IMG_WIDTH, IMG_HEIGHT, 1)

# Encode labels (e.g., Person_1, Person_2 to numerical labels)
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# One-hot encode the labels
y_onehot = to_categorical(y_encoded)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y_onehot, 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(64, (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(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

# Convolutional layer 3
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

# Convolutional layer 4 (new layer for additional complexity)
model.add(Conv2D(512, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten layer
model.add(Flatten())

# Dense layer
model.add(Dense(512, activation='relu'))  # Increased number of units for better
learning capacity
model.add(Dropout(0.5))  # Increased dropout to reduce overfitting

num_classes = len(np.unique(y_encoded))  # Number of unique persons in your
dataset
# Output layer (softmax for multi-class classification)
model.add(Dense(num_classes, activation='softmax'))
```

```
# Compile model
model.compile(optimizer=Adam(learning_rate=0.0001),
loss='categorical_crossentropy', metrics=['accuracy'])

# Define early stopping callback
early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=15,  # Increased patience to allow the model more time to improve
    restore_best_weights=True
)

# Fit the model
history = model.fit(
    datagen.flow(X_train, y_train, batch_size=32),
    validation_data=(X_test, y_test),
    epochs=200,
    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_person_mo
del.h5')

# Ensure that the y_test and y_pred are one-hot encoded or properly encoded
before applying argmax
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

# If label_encoder is not used, we'll define the class names based on unique labels
in the dataset
class_names = [f'Person_{i}' for i in range(1, num_classes + 1)]  # Assuming
`num_classes` gives the number of persons

# Print classification report
```

```
print(classification_report(y_true_classes, y_pred_classes,
target_names=class_names))

# Confusion Matrix
cm = confusion_matrix(y_true_classes, y_pred_classes)

# Plot Confusion Matrix
plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names,
yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Function to load and preprocess a single image
def preprocess_image(img_path):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (IMG_WIDTH, IMG_HEIGHT))  # Resize to match the
input dimensions
    img = img / 255.0  # Normalize pixel values to [0,1]
    img = img.reshape(1, IMG_WIDTH, IMG_HEIGHT, 1)  # Reshape to match
model input shape
    return img

# Path to the image you want to predict
image_path = '/content/drive/MyDrive/Sign_Data/Real/original_18_16.png'  #
Provide the path to your image

# Preprocess the image
single_image = preprocess_image(image_path)

# Predict the class
prediction = model.predict(single_image)

# Get the predicted class index
predicted_class_index = np.argmax(prediction)

# Assuming your class names are formatted as Person_1, Person_2, ...,
Person_num_classes
class_names = [f'Person_{i}' for i in range(1, num_classes + 1)]
```

```
# Get the predicted class name
predicted_class_name = class_names[predicted_class_index]

# Output the predicted class
print(f'Predicted class: {predicted_class_name}')

# Optionally, display the image and the prediction
plt.imshow(cv2.imread(image_path, cv2.IMREAD_GRAYSCALE), cmap='gray')
plt.title(f'Predicted: {predicted_class_name}')
plt.axis('off')
plt.show()
```

Model 2 -

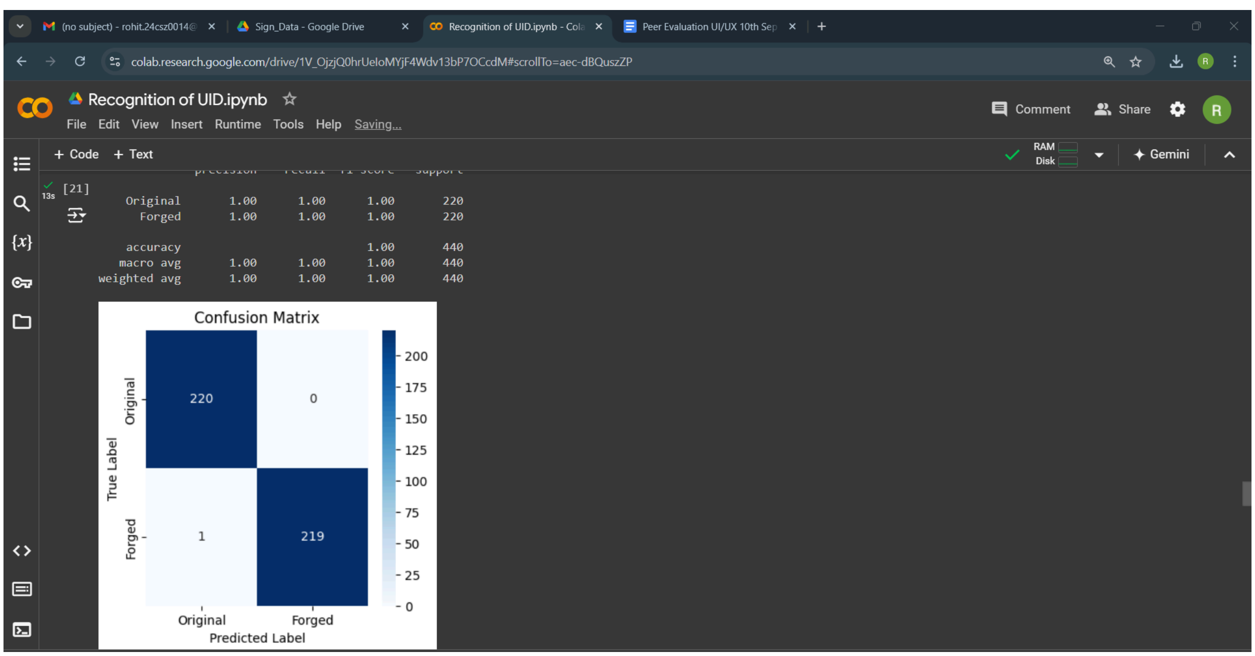
```
import os
import cv2
import numpy as np
import tensorflow as tf
from google.colab import drive
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

drive.mount('/content/drive')

# Define image dimensions
IMG_WIDTH = 256
IMG_HEIGHT = 256

# Load images from the dataset directory
def load_images(folder):
    images = []
    labels = []
    person_labels = {}
    signature_labels = {}  # To distinguish between 'real' and 'forged' signatures
```

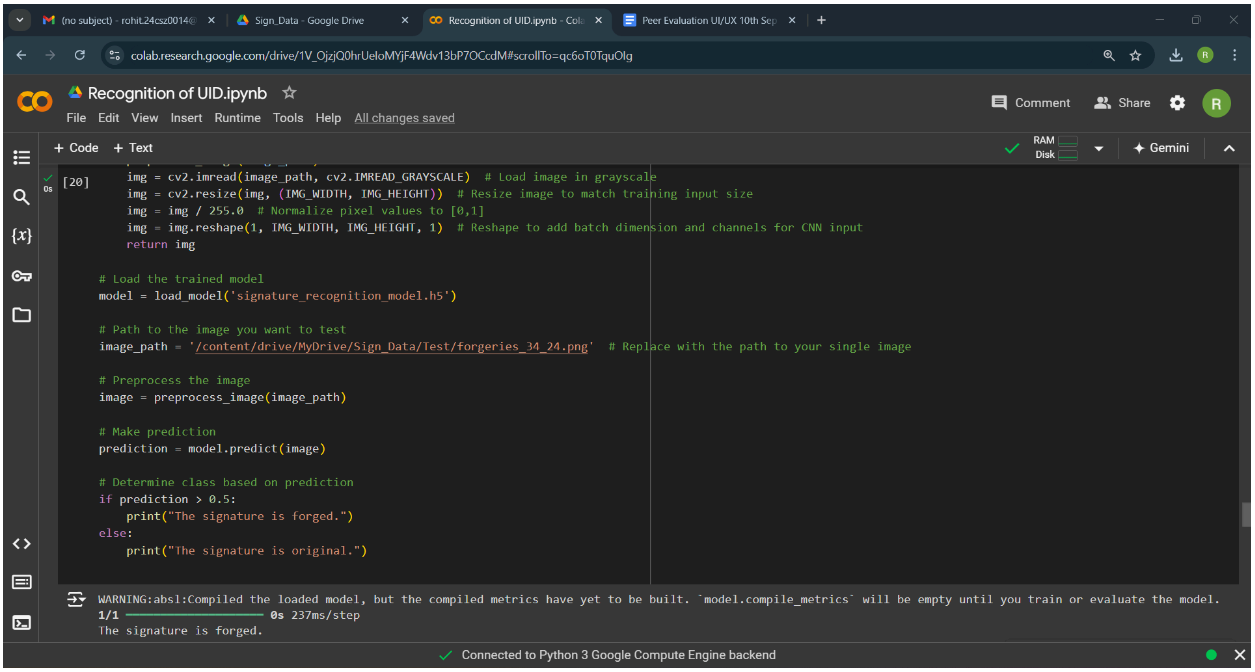
```
    # Iterate over files in the dataset folder
    for filename in os.listdir(folder):
        if filename.endswith('.jpg') or filename.endswith('.png'):  # Process image
files
            img_path = os.path.join(folder, filename)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            img = cv2.resize(img, (IMG_WIDTH, IMG_HEIGHT))
            img = img / 255.0  # Normalize pixel values to [0,1]
            img = np.expand_dims(img, axis=-1)  # Add channel dimension
            images.append(img)

            # Extract person number and signature type from filename
            parts = filename.split('_')
            person_id = parts[1]  # Person ID
            signature_type = parts[0]  # 'original' or 'forged'

            # Assign a unique label to each person
            if person_id not in person_labels:
                person_labels[person_id] = len(person_labels)

            # Assign a unique label to each signature type
            if signature_type not in signature_labels:
                signature_labels[signature_type] = len(signature_labels)

            # Create label combining person_id and signature_type
            label = person_labels[person_id] * len(signature_labels) +
signature_labels[signature_type]
            labels.append(label)

    # Convert lists to numpy arrays
    images = np.array(images)
    labels = np.array(labels)
    # Expand dimensions to include channel size (required for models expecting
RGB images)
    images = np.repeat(images, 3, axis=-1)

    return images, labels, len(person_labels), len(signature_labels)

# Define and build the model
def build_model(num_classes):
```

```
    base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=(IMG_WIDTH, IMG_HEIGHT, 3))
    base_model.trainable = False

    x = base_model.output
    x = Flatten()(x)
    x = Dense(512, activation='relu')(x)
    predictions = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=predictions)
    model.compile(optimizer=Adam(learning_rate=1e-4),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    return model

# Load dataset
dataset_folder = '/content/drive/MyDrive/Real'  # Update with your dataset folder
X, y, num_persons, num_signature_types = load_images(dataset_folder)

num_classes = num_persons * num_signature_types  # Total number of classes

# Build and print model summary
model = build_model(num_classes)
model.summary()

# Create data generators
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Split the dataset into training and validation sets (80/20 split)
split_index = int(0.8 * len(X))
X_train, X_val = X[:split_index], X[split_index:]
```

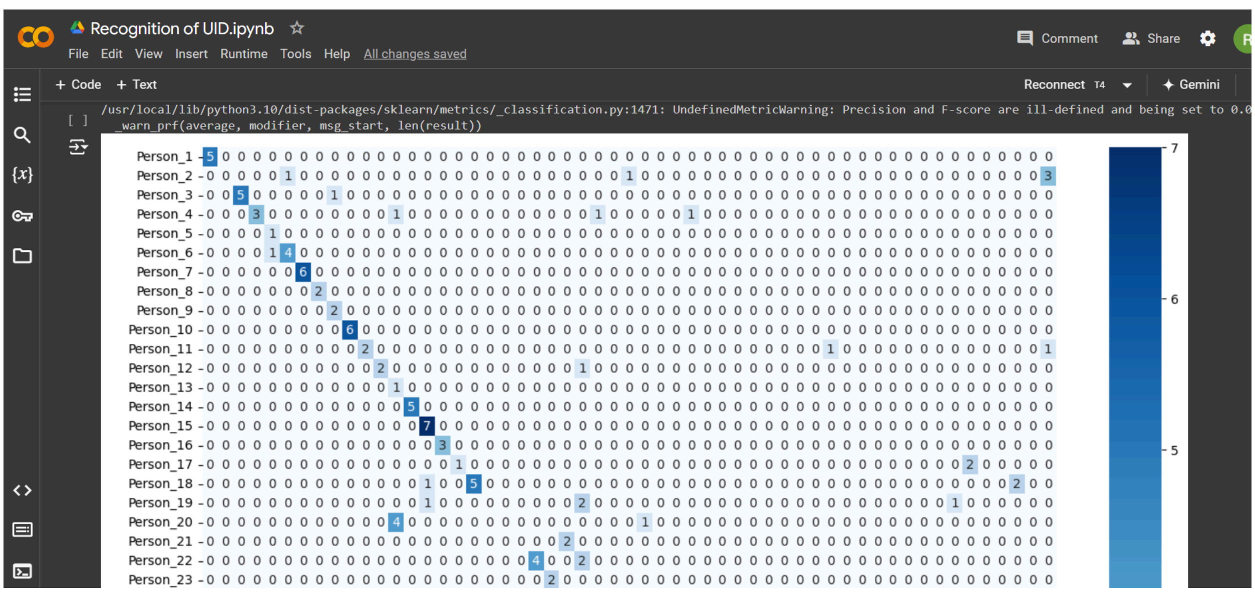
```
y_train, y_val = y[:split_index], y[split_index:]

# Create data generators from arrays
train_generator = train_datagen.flow(X_train, y_train, batch_size=32)
val_generator = train_datagen.flow(X_val, y_val, batch_size=32)

# Train the model
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=val_generator
)

# Save the model
model.save('signature_recognition_model.h5')
```

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

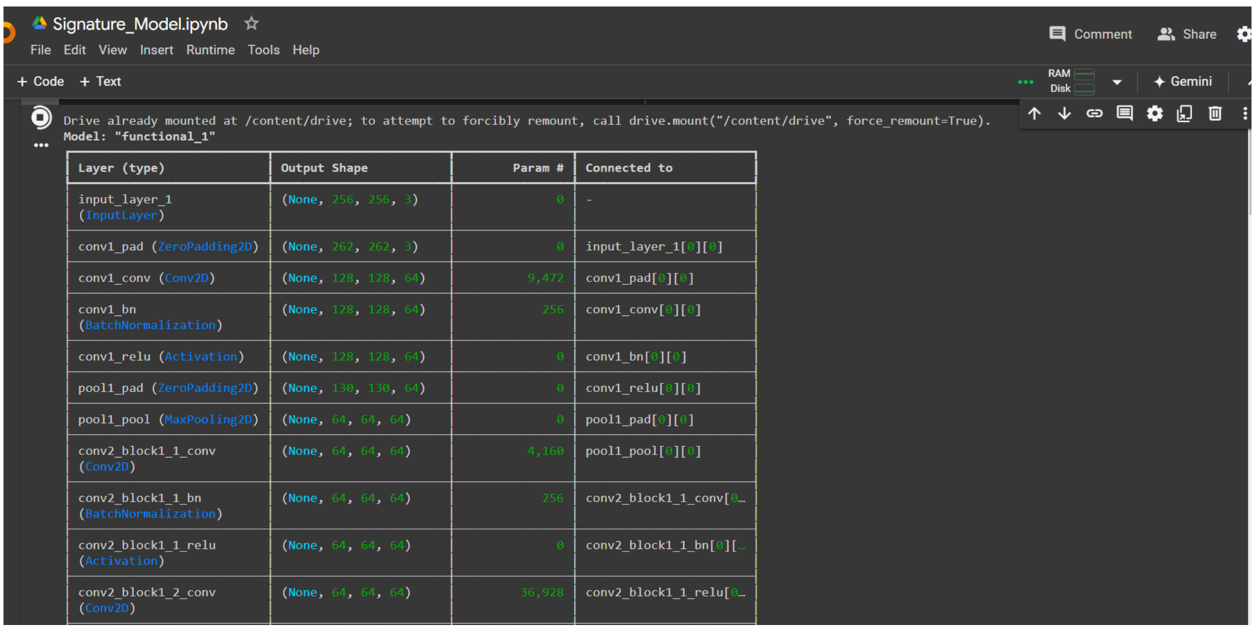




Recognition of UID.ipynb
File Edit View Insert Runtime Tools Help All changes saved
colab.research.google.com/drive/1V_OjzjQ0hrUeloMYjF4Wdv13bP7OCcdM#scrollTo=qc6oT0TquOlg
[20]
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE) # Load image in grayscale
    img = cv2.resize(img, (IMG_WIDTH, IMG_HEIGHT)) # 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.")
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
1/1 0s 237ms/step
The signature is forged.
Connected to Python 3 Google Compute Engine backend

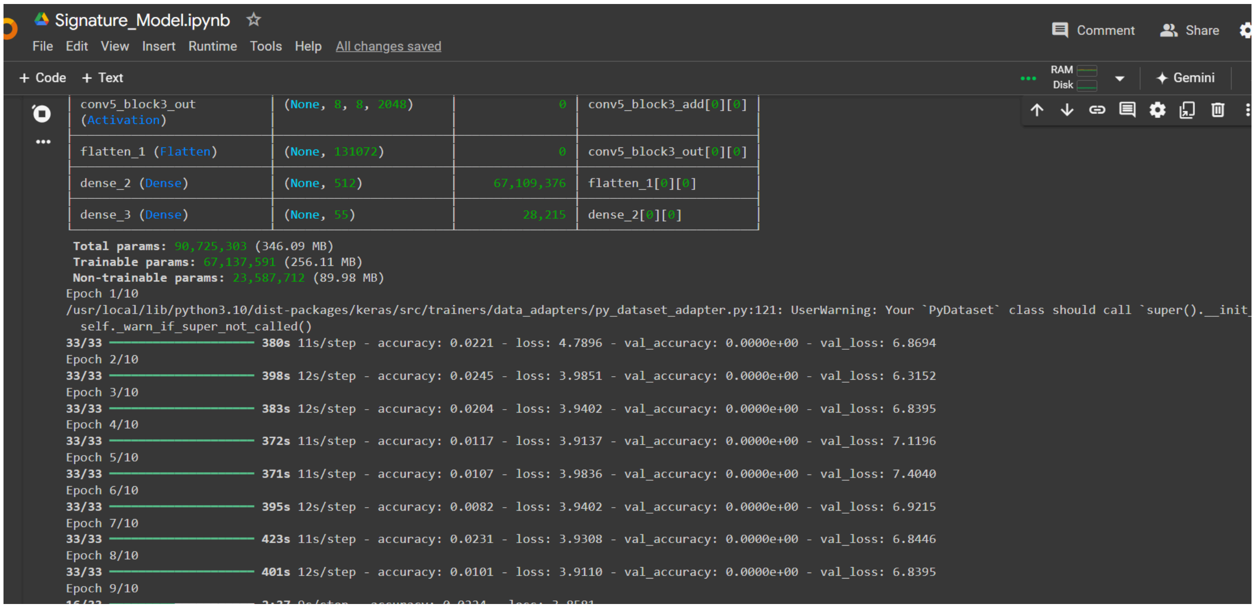


Recognition of UID.ipynb
File Edit View Insert Runtime Tools Help All changes saved
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
_warn_prf(average, modifier, msg_start, len(result))
Person_1
Person_2
Person_3
Person_4
Person_5
Person_6
Person_7
Person_8
Person_9
Person_10
Person_11
Person_12
Person_13
Person_14
Person_15
Person_16
Person_17
Person_18
Person_19
Person_20
Person_21
Person_22
Person_23
7
6
5




```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Model: "functional_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 256, 256, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 262, 262, 3)	0	input_layer_1[0][0]
conv1_conv (Conv2D)	(None, 128, 128, 64)	9,472	conv1_pad[0][0]
conv1_bn (BatchNormalization)	(None, 128, 128, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 128, 128, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 130, 130, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 64, 64, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 64, 64, 64)	4,160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormalization)	(None, 64, 64, 64)	256	conv2_block1_1_conv[0…
conv2_block1_1_relu (Activation)	(None, 64, 64, 64)	0	conv2_block1_1_bn[0][…
conv2_block1_2_conv (Conv2D)	(None, 64, 64, 64)	36,928	conv2_block1_1_relu[0…



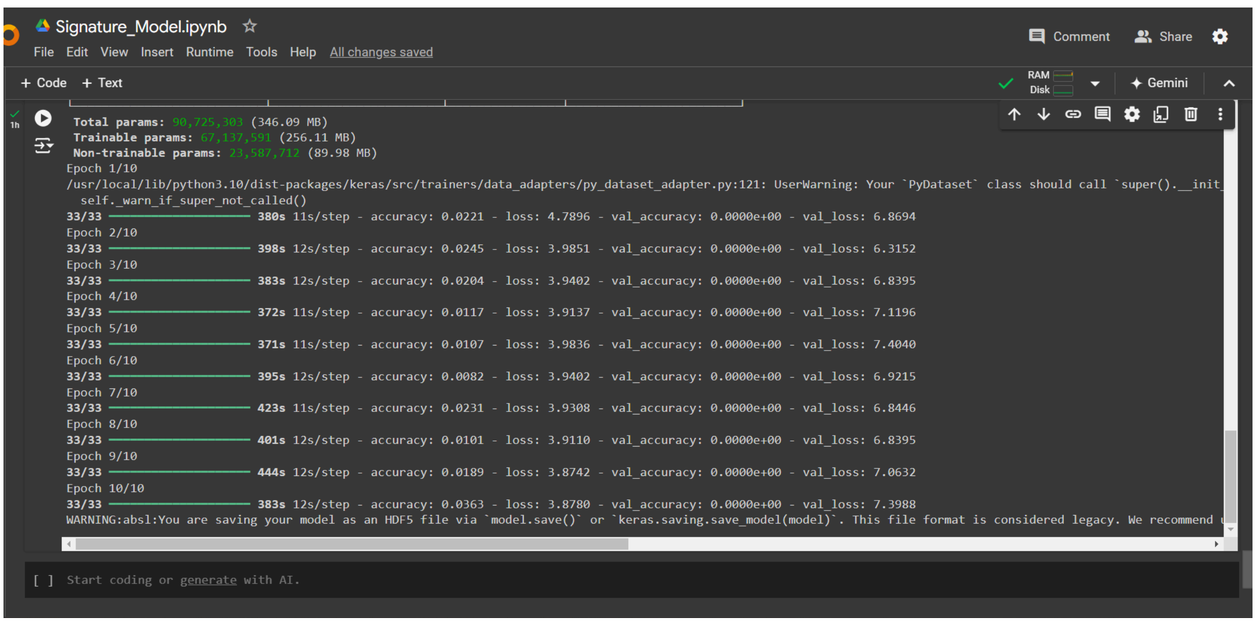
Signature_Model.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

conv5_block3_out (Activation)	(None, 8, 8, 2048)	0	conv5_block3_add[0][0]
flatten_1 (Flatten)	(None, 131072)	0	conv5_block3_out[0][0]
dense_2 (Dense)	(None, 512)	67,109,376	flatten_1[0][0]
dense_3 (Dense)	(None, 55)	28,215	dense_2[0][0]

```
 Total params: 90,725,303 (346.09 MB)
 Trainable params: 67,137,591 (256.11 MB)
 Non-trainable params: 23,587,712 (89.98 MB)
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init_
  self._warn_if_super_not_called()
33/33 ━━━━━━━━━━━━━━━━━━━━ 380s 11s/step - accuracy: 0.0221 - loss: 4.7896 - val_accuracy: 0.0000e+00 - val_loss: 6.8694
Epoch 2/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 398s 12s/step - accuracy: 0.0245 - loss: 3.9851 - val_accuracy: 0.0000e+00 - val_loss: 6.3152
Epoch 3/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 383s 12s/step - accuracy: 0.0204 - loss: 3.9402 - val_accuracy: 0.0000e+00 - val_loss: 6.8395
Epoch 4/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 372s 11s/step - accuracy: 0.0117 - loss: 3.9137 - val_accuracy: 0.0000e+00 - val_loss: 7.1196
Epoch 5/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 371s 11s/step - accuracy: 0.0107 - loss: 3.9836 - val_accuracy: 0.0000e+00 - val_loss: 7.4040
Epoch 6/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 395s 12s/step - accuracy: 0.0082 - loss: 3.9402 - val_accuracy: 0.0000e+00 - val_loss: 6.9215
Epoch 7/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 423s 11s/step - accuracy: 0.0231 - loss: 3.9308 - val_accuracy: 0.0000e+00 - val_loss: 6.8446
Epoch 8/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 401s 12s/step - accuracy: 0.0101 - loss: 3.9110 - val_accuracy: 0.0000e+00 - val_loss: 6.8395
Epoch 9/10
```



Signature_Model.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
 Total params: 90,725,303 (346.09 MB)
 Trainable params: 67,137,591 (256.11 MB)
 Non-trainable params: 23,587,712 (89.98 MB)
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init_
  self._warn_if_super_not_called()
33/33 ━━━━━━━━━━━━━━━━━━━━ 380s 11s/step - accuracy: 0.0221 - loss: 4.7896 - val_accuracy: 0.0000e+00 - val_loss: 6.8694
Epoch 2/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 398s 12s/step - accuracy: 0.0245 - loss: 3.9851 - val_accuracy: 0.0000e+00 - val_loss: 6.3152
Epoch 3/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 383s 12s/step - accuracy: 0.0204 - loss: 3.9402 - val_accuracy: 0.0000e+00 - val_loss: 6.8395
Epoch 4/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 372s 11s/step - accuracy: 0.0117 - loss: 3.9137 - val_accuracy: 0.0000e+00 - val_loss: 7.1196
Epoch 5/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 371s 11s/step - accuracy: 0.0107 - loss: 3.9836 - val_accuracy: 0.0000e+00 - val_loss: 7.4040
Epoch 6/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 395s 12s/step - accuracy: 0.0082 - loss: 3.9402 - val_accuracy: 0.0000e+00 - val_loss: 6.9215
Epoch 7/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 423s 11s/step - accuracy: 0.0231 - loss: 3.9308 - val_accuracy: 0.0000e+00 - val_loss: 6.8446
Epoch 8/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 401s 12s/step - accuracy: 0.0101 - loss: 3.9110 - val_accuracy: 0.0000e+00 - val_loss: 6.8395
Epoch 9/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 444s 12s/step - accuracy: 0.0189 - loss: 3.8742 - val_accuracy: 0.0000e+00 - val_loss: 7.0632
Epoch 10/10
33/33 ━━━━━━━━━━━━━━━━━━━━ 383s 12s/step - accuracy: 0.0363 - loss: 3.8780 - val_accuracy: 0.0000e+00 - val_loss: 7.3988
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend
```

[ ] Start coding or generate with AI.