In [ ]:

```
#Troy Krupinski
#tsk0064
#CSCE 5215
#10/20/2024
#Project 2
# Importing necessary libraries
import pandas as pd
import numpy as np
import os
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Input
from tensorflow.keras.optimizers import SGD, Adam, RMSprop
from sklearn.metrics import recall_score, f1_score, precision_score, confusi
from sklearn.utils import resample
import matplotlib.pyplot as plt
import seaborn as sns

# Paths to dataset (adjusted for your local environment)
base_path = "C:/Users/dunke/Desktop/New folder/CelebA/"
annotations_path = os.path.join(base_path, 'Anno')
eval_path = os.path.join(base_path, 'Eval')
images_path = os.path.join(base_path, 'Img/img_align_celeba')

# Load attributes (Male, Young, and Smiling)
def load_attributes():
    data_path = os.path.join(annotations_path, 'list_attr_celeba.txt')
    data = pd.read_csv(data_path, sep=r'\s+', skiprows=1)
    data = data.reset_index()
    data = data.rename(columns={'index': 'image_id'})
    return data[['image_id', 'Male', 'Young', 'Smiling']]

# Load the partition data
def load_partitions():
    partition_path = os.path.join(eval_path, 'list_eval_partition.txt')
    partition = pd.read_csv(partition_path, sep=r'\s+', header=None, names=[
    return partition

# Load landmarks (for mouth and eye width calculation)
def load_landmarks():
    landmarks_path = os.path.join(annotations_path, 'list_landmarks_align_ce
    landmarks = pd.read_csv(landmarks_path, sep=r'\s+', skiprows=1)
    landmarks = landmarks.reset_index()
    landmarks = landmarks.rename(columns={'index': 'image_id'})
    landmarks['mouth_width'] = landmarks['rightmouth_x'] - landmarks['leftmo
    landmarks['eye_width'] = landmarks['righteye_x'] - landmarks['lefteye_x'
    return landmarks[['image_id', 'mouth_width', 'eye_width']]

# Merge data
def merge_data(attributes, partition, landmarks):
    attributes['Male'] = attributes['Male'].replace({-1: 0})
    attributes['Young'] = attributes['Young'].replace({-1: 0})
```

```
    attributes['Smiling'] = attributes['Smiling'].replace({-1: 0})
    df = pd.merge(attributes, partition, on='image_id', how='inner')
    df = pd.merge(df, landmarks, on='image_id', how='inner')
    return df

# Split data into training, validation, and test sets
def split_data(df):
    train_df = df[df['partition'] == 0]
    test_df = df[df['partition'] == 1]
    val_df = df[df['partition'] == 2]
    return train_df, test_df, val_df

# Sample data to reduce training time
def sample_data(train_df, val_df, test_df, train_size=7500, val_size=750, te
    train_df = train_df.sample(n=train_size, random_state=42)
    val_df = val_df.sample(n=val_size, random_state=42)
    test_df = test_df.sample(n=test_size, random_state=42)
    return train_df, val_df, test_df

# Data generator for image batches
class DataGenerator(tf.keras.utils.Sequence):
    def __init__(self, df, batch_size=64, dim=(128, 128), n_channels=3, targ
        self.dim = dim
        self.batch_size = batch_size
        self.df = df
        self.n_channels = n_channels
        self.targets = targets
        self.shuffle = shuffle
        self.on_epoch_end()

    def __len__(self):
        return int(np.ceil(len(self.df) / self.batch_size))

    def __getitem__(self, index):
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_si
        df_temp = self.df.iloc[indexes].reset_index(drop=True)
        X, y = self.__data_generation(df_temp)
        return X, y

    def on_epoch_end(self):
        self.indexes = np.arange(len(self.df))
        if self.shuffle:
            np.random.shuffle(self.indexes)

    def __data_generation(self, df_temp):
        X = np.empty((len(df_temp), *self.dim, self.n_channels))
        y = np.empty((len(df_temp), 1), dtype=int)  # Ensure we return 1D ar

        for i, row in df_temp.iterrows():
            img_path = os.path.join(images_path, row['image_id'])
            img = image.load_img(img_path, target_size=self.dim)
            img = image.img_to_array(img)
            img /= 255.0
            X[i,] = img
            y[i,] = row[self.targets[0]]  # Only use the first target (e.g.,
```

```python
        return X, y


# R1 Gender Classification Model
def build_model_gender(input_shape):
    vgg16 = VGG16(input_shape=input_shape, include_top=False, weights='image
    vgg16.trainable = False  # Freeze the VGG16 layers

    inputs = Input(shape=input_shape)
    x = vgg16(inputs, training=False)
    x = Flatten()(x)
    x = Dense(512, activation='relu')(x)
    gender_output = Dense(1, activation='sigmoid', name='gender')(x)  # Sigm

    model = Model(inputs=inputs, outputs=[gender_output])
    model.compile(optimizer=SGD(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model


# Ensure the model outputs both gender and age predictions
#OLD MODEL FOR R2, DEPRECATED / REPLACED BY build_model_multiclass

def build_model_multitarget(input_shape):
    vgg16 = VGG16(input_shape=input_shape, include_top=False, weights='image
    for layer in vgg16.layers[:15]:
        layer.trainable = False  # Keep lower layers frozen

    inputs = Input(shape=input_shape)
    x = vgg16(inputs, training=True)
    x = Flatten()(x)
    x = Dense(512, activation='relu')(x)

    # Output for gender (Male/Female)
    gender_output = Dense(1, activation='sigmoid', name='gender')(x)

    # Output for age (Young/Old)
    age_output = Dense(1, activation='sigmoid', name='age')(x)

    model = Model(inputs=inputs, outputs=[gender_output, age_output])
    model.compile(optimizer=SGD(learning_rate=0.001),
                  loss={'gender': 'binary_crossentropy', 'age': 'binary_cros
                  metrics={'gender': ['accuracy', 'precision'], 'age': ['acc

    return model

#

# Preprocess the CelebA dataset with landmarks
def preprocess_celeba_dataset():
    attributes = load_attributes()
    partition = load_partitions()
    landmarks = load_landmarks()
    df = merge_data(attributes, partition, landmarks)
```

```
    train_df, val_df, test_df = split_data(df)
    train_df, val_df, test_df = sample_data(train_df, val_df, test_df, trair
    return train_df, val_df, test_df

# R1: Train and evaluate gender classification model
def r1_gender_classification():
    train_df, val_df, test_df = preprocess_celeba_dataset()

    # Initialize data generators
    train_generator = DataGenerator(train_df, batch_size=64, dim=(128, 128),
    val_generator = DataGenerator(val_df, batch_size=64, dim=(128, 128), n_c
    test_generator = DataGenerator(test_df, batch_size=64, dim=(128, 128), r

    # Build model
    model = build_model_gender(input_shape=(128, 128, 3))

    # Fit the model
    model.fit(train_generator, epochs=3, validation_data=val_generator)

    # Evaluate model on the test set
    test_loss, test_accuracy = model.evaluate(test_generator)
    print(f"R1 (Gender Classification) Results - Loss: {test_loss}, Accuracy

    # Make predictions on the test set
    predictions = model.predict(test_generator)
    predicted_labels = np.where(predictions > 0.5, 1, 0)  # Convert probabil

    # Extract true labels
    true_labels = test_df['Male'].values  # Ensure the labels are binary (0

    # Confusion Matrix for Gender Classification
    cm_gender = confusion_matrix(true_labels, predicted_labels)

    # Plot confusion matrix
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm_gender, annot=True, fmt="d", cmap="Blues")
    plt.title('Confusion Matrix for Gender Classification (R1)')
    plt.ylabel('Actual Label')
    plt.xlabel('Predicted Label')
    plt.show()

    return test_loss, test_accuracy




# R2: Train and evaluate gender and age classification model
def r2_gender_age_classification():
    train_df, val_df, test_df = preprocess_celeba_dataset()

    # Create a new target column for 4-class combinations
    train_df['class'] = train_df.apply(lambda row: f"{row['Male']}_{row['You
    val_df['class'] = val_df.apply(lambda row: f"{row['Male']}_{row['Young']
    test_df['class'] = test_df.apply(lambda row: f"{row['Male']}_{row['Young

    # Map the class to a unique integer
```

```python
    class_mapping = {"1_1": 0, "1_0": 1, "0_1": 2, "0_0": 3}
    train_df['class'] = train_df['class'].map(class_mapping)
    val_df['class'] = val_df['class'].map(class_mapping)
    test_df['class'] = test_df['class'].map(class_mapping)

    # Initialize data generators with multi-class target
    train_generator = DataGenerator(train_df, batch_size=64, dim=(128, 128),
    val_generator = DataGenerator(val_df, batch_size=64, dim=(128, 128), n_c
    test_generator = DataGenerator(test_df, batch_size=64, dim=(128, 128), r

    # Build multi-class classification model
    model = build_model_multiclass(input_shape=(128, 128, 3))

    # Train the model
    model.fit(train_generator, epochs=3, validation_data=val_generator)

    # Evaluate the model on the test set
    test_loss, test_accuracy = model.evaluate(test_generator)
    print(f"R2 (Gender and Age Classification) Results - Loss: {test_loss},

    # Make predictions on the test set
    predictions = model.predict(test_generator)
    predicted_classes = np.argmax(predictions, axis=1)

    # Extract true labels
    true_classes = test_df['class'].values

    # Confusion Matrix for 4-Class Combinations
    cm = confusion_matrix(true_classes, predicted_classes)

    # Plot confusion matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Young M
                yticklabels=["Young Male", "Old Male", "Young Female", "Old
    plt.title('Confusion Matrix for Gender and Age Classification (R2)')
    plt.ylabel('Actual Label')
    plt.xlabel('Predicted Label')
    plt.show()

    # Calculate per-class accuracy
    correct_per_class = cm.diagonal()
    total_per_class = cm.sum(axis=1)
    accuracy_per_class = correct_per_class / total_per_class

    # Print accuracy results
    class_labels = ["Young Male", "Old Male", "Young Female", "Old Female"]
    for i, accuracy in enumerate(accuracy_per_class):
        print(f"Accuracy for {class_labels[i]}: {accuracy:.2f} ({correct_per

    return test_loss, test_accuracy  # Return two values, combined loss and


# Build multi-class classification model
def build_model_multiclass(input_shape):
    vgg16 = VGG16(input_shape=input_shape, include_top=False, weights='image
    vgg16.trainable = False  # Freeze the VGG16 layers
```

```
    inputs = Input(shape=input_shape)
    x = vgg16(inputs, training=False)
    x = Flatten()(x)
    x = Dense(512, activation='relu')(x)
    output = Dense(4, activation='softmax')(x)  # 4 classes: Young Male, Old

    model = Model(inputs=inputs, outputs=[output])
    model.compile(optimizer=SGD(learning_rate=0.001),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model




def build_model_smiling(input_shape):
    vgg16 = VGG16(input_shape=input_shape, include_top=False, weights='image
    vgg16.trainable = False  # Freeze the VGG16 layers

    inputs = Input(shape=input_shape)
    x = vgg16(inputs, training=False)
    x = Flatten()(x)
    x = Dense(512, activation='relu')(x)
    smiling_output = Dense(1, activation='sigmoid', name='smiling')(x)

    model = Model(inputs=inputs, outputs=[smiling_output])
    model.compile(optimizer=SGD(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['accuracy', tf.keras.metrics.Precision(name='prec

    return model


# R3: Mouth Width and Eye Width Classification with Confusion Matrices

def r3_mouth_and_eye_classification():
    # Preprocess the dataset
    train_df, val_df, test_df = preprocess_celeba_dataset()

    # Quartile Calculation for mouth width and eye width
    train_df['mouth_width_q'] = pd.qcut(train_df['mouth_width'], 4, labels=[
    train_df['eye_width_q'] = pd.qcut(train_df['eye_width'], 4, labels=[1, 2

    # (a) Train models for Q1 and Q4 for mouth width (Smiling)
    print("\n=== (a) Mouth Width Quartile 1 (Q1) vs Quartile 4 (Q4) ===")
    q1_train = train_df[train_df['mouth_width_q'] == 1]
    q4_train = train_df[train_df['mouth_width_q'] == 4]

    # Balance data if necessary (example with Q1)
    print("Before balancing Q1 dataset: ", q1_train['Smiling'].value_counts(
    q1_smiling_minority = q1_train[q1_train['Smiling'] == 1]
    q1_not_smiling_majority = q1_train[q1_train['Smiling'] == 0]
    q1_smiling_upsampled = resample(q1_smiling_minority,
```

```
                                replace=True,
                                n_samples=len(q1_not_smiling_majority),
                                random_state=42)
q1_train_balanced = pd.concat([q1_not_smiling_majority, q1_smiling_upsam
print("After balancing Q1 dataset: ", q1_train_balanced['Smiling'].value

# Repeat balancing for Q4 if necessary
print("Before balancing Q4 dataset: ", q4_train['Smiling'].value_counts(
q4_smiling_minority = q4_train[q4_train['Smiling'] == 1]
q4_not_smiling_majority = q4_train[q4_train['Smiling'] == 0]
q4_smiling_upsampled = resample(q4_smiling_minority,
                                replace=True,
                                n_samples=len(q4_not_smiling_majority),
                                random_state=42)
q4_train_balanced = pd.concat([q4_not_smiling_majority, q4_smiling_upsam
print("After balancing Q4 dataset: ", q4_train_balanced['Smiling'].value

model_q1 = build_model_smiling(input_shape=(128, 128, 3))
model_q4 = build_model_smiling(input_shape=(128, 128, 3))

train_q1_generator = DataGenerator(q1_train_balanced, batch_size=64, dim
train_q4_generator = DataGenerator(q4_train_balanced, batch_size=64, dim
test_generator = DataGenerator(test_df, batch_size=64, dim=(128, 128), r

# Using Adam optimizer for better convergence
model_q1.compile(optimizer='adam', loss='binary_crossentropy', metrics=[
model_q4.compile(optimizer='adam', loss='binary_crossentropy', metrics=[

model_q1.fit(train_q1_generator, epochs=3)
model_q4.fit(train_q4_generator, epochs=3)

# Evaluate models on the test set and print confusion matrices
threshold = 0.5  # Adjust this as needed
q1_predictions = np.where(model_q1.predict(test_generator) > threshold,
q4_predictions = np.where(model_q4.predict(test_generator) > threshold,

test_smiling_labels = test_df['Smiling'].values

# (b) Compute precision, recall, and F1 scores for "Smiling"
print("\n=== (b) Mouth Width Sensitivity Analysis ===")

P1 = precision_score(test_smiling_labels, q1_predictions, zero_division=
P3 = precision_score(test_smiling_labels, q4_predictions, zero_division=
R1 = recall_score(test_smiling_labels, q1_predictions, zero_division=1)
R3 = recall_score(test_smiling_labels, q4_predictions, zero_division=1)
F1_q1 = f1_score(test_smiling_labels, q1_predictions, zero_division=1)
F1_q4 = f1_score(test_smiling_labels, q4_predictions, zero_division=1)

print(f"Precision for Q1 model (Smiling): {P1}")
print(f"Recall for Q1 model (Smiling): {R1}")
print(f"F1 Score for Q1 model (Smiling): {F1_q1}")
print(f"Precision for Q4 model (Smiling): {P3}")
print(f"Recall for Q4 model (Smiling): {R3}")
print(f"F1 Score for Q4 model (Smiling): {F1_q4}")

# Confusion matrix for Q1
```

```python
    q1_cm = confusion_matrix(test_smiling_labels, q1_predictions)
    print("\nConfusion Matrix for Q1 (Smiling):")
    sns.heatmap(q1_cm, annot=True, fmt="d", cmap="Blues")
    plt.title('Confusion Matrix for Q1 (Smiling)')
    plt.show()

    # Confusion matrix for Q4
    q4_cm = confusion_matrix(test_smiling_labels, q4_predictions)
    print("\nConfusion Matrix for Q4 (Smiling):")
    sns.heatmap(q4_cm, annot=True, fmt="d", cmap="Blues")
    plt.title('Confusion Matrix for Q4 (Smiling)')
    plt.show()

    # (c) Train models for Q1 and Q4 for eye width (Female classification)
    print("\n=== (c) Eye Width Quartile 1 (Q1) vs Quartile 4 (Q4) for Female
    q1_train_female = train_df[(train_df['eye_width_q'] == 1) & (train_df['M
    q4_train_female = train_df[(train_df['eye_width_q'] == 4) & (train_df['M

    # Balance Q1 and Q4 female data if necessary (similar to above)

    model_q1_female = build_model_smiling(input_shape=(128, 128, 3))
    model_q4_female = build_model_smiling(input_shape=(128, 128, 3))

    train_q1_female_generator = DataGenerator(q1_train_female, batch_size=64
    train_q4_female_generator = DataGenerator(q4_train_female, batch_size=64

    model_q1_female.compile(optimizer='adam', loss='binary_crossentropy', me
    model_q4_female.compile(optimizer='adam', loss='binary_crossentropy', me

    model_q1_female.fit(train_q1_female_generator, epochs=3)
    model_q4_female.fit(train_q4_female_generator, epochs=3)

    # Evaluate female models on the test set and print confusion matrices
    q1_female_predictions = np.where(model_q1_female.predict(test_generator)
    q4_female_predictions = np.where(model_q4_female.predict(test_generator)

    # (d) Compute precision, recall, and F1 scores for "Smiling" in females
    print("\n=== (d) Eye Width Sensitivity Analysis for Female Classificatio

    P1_female = precision_score(test_smiling_labels, q1_female_predictions,
    P3_female = precision_score(test_smiling_labels, q4_female_predictions,
    R1_female = recall_score(test_smiling_labels, q1_female_predictions, zer
    R3_female = recall_score(test_smiling_labels, q4_female_predictions, zer
    F1_q1_female = f1_score(test_smiling_labels, q1_female_predictions, zero
    F1_q4_female = f1_score(test_smiling_labels, q4_female_predictions, zero

    print(f"Precision for Q1 female model (Smiling): {P1_female}")
    print(f"Recall for Q1 female model (Smiling): {R1_female}")
    print(f"F1 Score for Q1 female model (Smiling): {F1_q1_female}")
    print(f"Precision for Q4 female model (Smiling): {P3_female}")
    print(f"Recall for Q4 female model (Smiling): {R3_female}")
    print(f"F1 Score for Q4 female model (Smiling): {F1_q4_female}")

    # Confusion matrix for Q1 (Female)
    q1_female_cm = confusion_matrix(test_smiling_labels, q1_female_predictio
    print("\nConfusion Matrix for Q1 (Female Smiling):")
```

```python
    sns.heatmap(q1_female_cm, annot=True, fmt="d", cmap="Blues")
    plt.title('Confusion Matrix for Q1 (Female Smiling)')
    plt.show()

    # Confusion matrix for Q4 (Female)
    q4_female_cm = confusion_matrix(test_smiling_labels, q4_female_predictio
    print("\nConfusion Matrix for Q4 (Female Smiling):")
    sns.heatmap(q4_female_cm, annot=True, fmt="d", cmap="Blues")
    plt.title('Confusion Matrix for Q4 (Female Smiling)')
    plt.show()

# Run the classification process for R3 and display confusion matrices
if __name__ == "__main__":
    print("=== Running R1 (Gender Classification) ===")
    r1_loss, r1_accuracy = r1_gender_classification()

    print("\n=== Running R2 (Gender and Age Classification) ===")
    r2_combined_loss, r2_overall_accuracy = r2_gender_age_classification()

    print("\n=== Running R3 (Mouth Width and Eye Width Classification) ===")
    r3_mouth_and_eye_classification()

    # Final results summary
    print("\n=== Final Results Summary ===")
    print(f"Results for R1 (Gender Classification):")
    print(f"Gender Accuracy: {r1_accuracy}, Gender Loss: {r1_loss}")

    print(f"\nResults for R2 (Gender and Age Classification):")
    print(f"Combined Loss: {r2_combined_loss}, Overall Accuracy: {r2_overall
```