Emre Zeytinoğlu 28190

CS 412 HW4

Google collab link: https://colab.research.google.com/drive/1nt2byllrUOsel-TmlWa8p1Gdb8sU4q0a?usp=sharing

```
[1]
# load data
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

After Google Drive is installed , we move on to import the essential libraries and modules for our gender classification work.

```
data = pd.read_csv('/content/drive/My Drive/celeba_30k.csv') # ent
data.head()
```

	image_id	Male	Blond_Hair	Eyeglasses	Wearing_Earrings	Bangs	Youn
0	000001.jpg	0	0	0	1	0	[illegible]
1	000002.jpg	0	0	0	0	0	[illegible]
2	000003.jpg	1	0	0	0	0	[illegible]
3	000004.jpg	0	0	0	1	0	[illegible]
4	000005.jpg	0	0	0	0	0	[illegible]

Next, we proceed with loading the CSV file containing the dataset.

```
[4]
gender_data = data[['image_id', 'Male']].copy()
gender_data.head()
```

	image_id	Male
0	000001.jpg	0
1	000002.jpg	0
2	000003.jpg	1
3	000004.jpg	0
4	000005.jpg	0

We choose the appropriate columns that contain the image IDs and gender labels in order to prepare the dataset for gender categorization.

```
[5]
#this will extract the contents of the zip file into a folder named
#do not extract the zip into your google drive (i.e don't use driv
#only change the left path

!unzip "/content/drive/My Drive/celeba_30k.zip" -d "/content/data"
```

```
inflating: /content/data/celeba_30k/015963.jpg
inflating: /content/data/__MACOSX/celeba_30k/._015963.jpg
inflating: /content/data/celeba_30k/023209.jpg
inflating: /content/data/__MACOSX/celeba_30k/._023209.jpg
inflating: /content/data/celeba_30k/024200.jpg
inflating: /content/data/__MACOSX/celeba_30k/._024200.jpg
```

It is necessary to unzip the provided zip file in order to access the image data for the CelebA dataset.

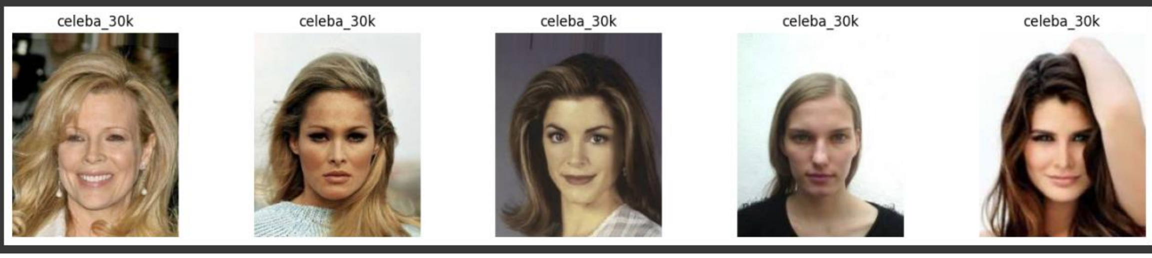
```
[6]
first_image_path = os.path.join("/content/data/celeba_30k/", gender
img = Image.open(first_image_path)
```

```
img
```

We use the Image module to open and display the first image from the dataset in order to visualize it. We use the Image module to open and display the first image from the dataset in order to visualize it.

```python
for i, (image_path, label) in enumerate(random_images):
    image = Image.open(image_path)
    axes[i].imshow(image)
    axes[i].set_title(label)
    axes[i].axis('off')

plt.tight_layout()
plt.show()
```



By running this code, we navigate to the downloaded photos' directory path and compile a list of all the image files' paths and labels. Then, we present five image-label combinations that were randomly chosen from the dataset.

```python
val_datagen = ImageDataGenerator() #augmentations for validation s
val_generator = val_datagen.flow_from_dataframe(
    val_df,
    data_path,
    x_col='image_id',
    y_col='Male',
    target_size=(224,224),
    class_mode='binary',
    batch_size=batch_size
)
```

```
Found 24000 validated image filenames belonging to 2 classes.
Found 3000 validated image filenames belonging to 2 classes.
```

We produce distinct data generators for the training and validation sets by running this code. We may preprocess the photos and enrich the data using the ImageDataGenerator objects train_datagen and val_datagen.

```
from keras.applications.vgg16 import VGG16

base_model = VGG16(weights='imagenet', input_shape = (224,224,3), i
base_model.summary()

```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-a
58889256/58889256 [==============================] - 0s 0us/step
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168
```

By running this code, we import the Keras library's VGG-16 model and build the base model.

```python
    return model

# Load the pre-trained VGG16 model
base_model = VGG16(weights='imagenet', input_shape=(224, 224, 3),

# Set the trainable attribute of all layers in the base model to F
for layer in base_model.layers:
    layer.trainable = False

# Create the gender classification model
model = gender_model(base_model)

# Print the model summary
model.summary()

```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 224, 224, 3)]     0

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 64)                1605696

 dense_1 (Dense)             (None, 1)                 65
```

The pre-trained VGG16 base model receives an input of shape (224, 224, 3) and generates a feature map with shape (7, 7, 512).
There are 16,322,449 total parameters in the model, of which 1,605,761 are trainable.
The pre-trained VGG16 base model's 14,716,688 extra parameters are not trainable.

```
    batch_size=batch_size,
    class_mode='binary'
)


# Compile the model



optimizer = SGD(learning_rate=0.001)  # Example optimizer, you can
loss = BinaryCrossentropy()  # Example loss function, you can choo

model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy']


# Train the model using generators with the fit method
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=val_generator,
    validation_steps=val_generator.samples // batch_size,
    workers=workers
)

```

```
Found 24000 validated image filenames belonging to 2 classes.
Found 3000 validated image filenames belonging to 2 classes.
```

Epoch 1/10
750/750 [==============================] - 295s 367ms/step - loss: 0.4627 - accuracy: 0.7758 - val_loss: 0.2830 - val_accuracy: 0.8901

Epoch 2/10
750/750 [==============================] - 274s 362ms/step - loss: 0.3551 - accuracy: 0.8435 - val_loss: 0.2923 - val_accuracy: 0.8723

Epoch 3/10
750/750 [==============================] - 276s 364ms/step - loss: 0.3192 - accuracy: 0.8642 - val_loss: 0.2422 - val_accuracy: 0.9002

Epoch 4/10
750/750 [==============================] - 275s 364ms/step - loss: 0.3018 - accuracy: 0.8713 - val_loss: 0.2493 - val_accuracy: 0.8962

Epoch 5/10
750/750 [==============================] - 281s 372ms/step - loss: 0.2906 - accuracy: 0.8760 - val_loss: 0.2224 - val_accuracy: 0.9120

Epoch 6/10

750/750 [================================] - 280s 370ms/step - loss: 0.2832 - accuracy: 0.8814 - val_loss: 0.2150 - val_accuracy: 0.9143

Epoch 7/10
750/750 [================================] - 282s 373ms/step - loss: 0.2768 - accuracy: 0.8832 - val_loss: 0.2676 - val_accuracy: 0.8921

Epoch 8/10
750/750 [================================] - 278s 366ms/step - loss: 0.2725 - accuracy: 0.8860 - val_loss: 0.2249 - val_accuracy: 0.9083

Epoch 9/10
750/750 [================================] - 279s 369ms/step - loss: 0.2647 - accuracy: 0.8884 - val_loss: 0.2186 - val_accuracy: 0.9106

Epoch 10/10
750/750 [================================] - 282s 371ms/step - loss: 0.2612 - accuracy: 0.8913 - val_loss: 0.2041 - val_accuracy: 0.9190

On the training set, the model had an accuracy of about 89-91%. On the validation set, it had an accuracy of about 87-92%. Over the epochs, the loss decreased, showing that the model's performance had improved.