1. Installing dependencies and setup

In [ ]:
```
!pip install --upgrade tensorflow[and-cuda] opencv-python matplotlib
```

2. Checking installations

In [ ]:
```
!pip list
```

3. Importing libraries

In [ ]:
```
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"  # Disables GPU
import tensorflow as tf
import cv2
import imghdr
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy
import cv2
from tensorflow.keras.models import load_model
```

4. Removing dodgy images from data sets

In [ ]:
```
data_dir = 'data'
```

In [ ]:
```
image_exts = ['jpeg','jpg', 'bmp', 'png']
```

5. Loading data

In [ ]:
```
data = tf.keras.utils.image_dataset_from_directory('data')
```

In [ ]:
```
data_iterator = data.as_numpy_iterator()
```

In [ ]:
```
batch = data_iterator.next()
```

In [ ]:
```
#checking what is assigned to 0 and 1

fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx, img in enumerate(batch[0][:4]):
    ax[idx].imshow(img.astype(int))
```

```
    ax[idx].title.set_text(batch[1][idx])

#happy = 0
#sad = 1
```

6. Scaling Data from 0-255 to 0-1

In [ ]:
```
data = data.map(lambda x,y: (x/255, y))
```

In [ ]:
```
data.as_numpy_iterator().next()
```

7. Splitting data

In [ ]:
```
len(data)
```

In [ ]:
```
train_size = int(len(data)*.8)
val_size = int(len(data)*.1)
test_size = int(len(data)*.1)
```

In [ ]:
```
print(train_size)
print(val_size)
print(test_size)
```

In [ ]:
```
train = data.take(train_size) #take 8 batches
val = data.skip(train_size).take(val_size) #skip batches already taken and take 1 b
test = data.skip(train_size+val_size).take(test_size) #skip batches already taken a
```

8. Building Model

In [ ]:
```
model = Sequential()
```

In [ ]:
```
tf.random.set_seed(1234)

model = Sequential(
    [
        ### START CODE HERE ###
        Conv2D(16, (3,3), strides=1, activation='relu', input_shape=(256,256,3), na
        MaxPooling2D(name="maxpool1"),

        Conv2D(32, (3,3), strides=1, activation='relu', name="conv2"),
        MaxPooling2D(name="maxpool2"),

        Conv2D(16, (3,3), strides=1, activation='relu', name="conv3"),
        MaxPooling2D(name="maxpool3"),

        Flatten(name="flatten"),
        Dense(256, activation='relu', name="dense1"),
        Dense(1, activation='sigmoid', name="output")
```

```
    ], name="my_model"
)
```

In [ ]:
```
model.compile('adam', loss=tf.losses.BinaryCrossentropy(), metrics=['accuracy'])
```

In [ ]:
```
model.summary()
```

9. Training

In [ ]:
```
logdir='logs'
```

In [ ]:
```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

In [ ]:
```
hist = model.fit(train, epochs=20, validation_data=val, callbacks=[tensorboard_call
```

10. Plotting

In [ ]:
```
fig = plt.figure()
plt.plot(hist.history['loss'], color='blue', label='loss')
plt.plot(hist.history['val_loss'], color='red', label='val_loss')
fig.suptitle('Loss', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```

In [ ]:
```
fig = plt.figure()
plt.plot(hist.history['accuracy'], color='blue', label='accuracy')
plt.plot(hist.history['val_accuracy'], color='red', label='val_accuracy')
fig.suptitle('Accuracy', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```

11. Evaluating Performance

In [303…
```
precision = Precision()
recall = Recall()
accuracy = BinaryAccuracy()

# Ensure dataset iteration does not go out of range
for batch in test.take(len(test)):  # Use `.take()` to iterate safely
    X, y = batch
    yhat = model.predict(X)

    precision.update_state(y, yhat)
    recall.update_state(y, yhat)
    accuracy.update_state(y, yhat)

print(f'Precision: {precision.result().numpy()}, Recall: {recall.result().numpy()},
```

```
print("yhat shape:", yhat.shape)
```

```
2025-01-20 22:37:04.601705: W tensorflow/core/lib/png/png_io.cc:89] PNG warning: iCC
P: known incorrect sRGB profile
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 198ms/step
Precision: 1.0, Recall: 1.0, Accuracy: 1.0
yhat shape: (32, 1)
```

In [304…
```
img = cv2.imread('happyExample2.jpg')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
```

In [ ]:
```
resizedImg = tf.image.resize(img, (256,256))
plt.imshow(resizedImg.numpy().astype(int))
plt.show()
```

In [ ]:
```
np.expand_dims(resizedImg, 0)
```

In [307…
```
yhat = model.predict(np.expand_dims(resizedImg/255, 0))
print(yhat)

if yhat > 0.5:
    print("Prediction: Sad")
else:
    print("Prediction: Happy")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step
[[0.01000653]]
Prediction: Happy
```

In [308…
```
img = cv2.imread('sadExample1.jpg')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

```
plt.show()
```

In [ ]:
```
resizedImg = tf.image.resize(img, (256,256))
plt.imshow(resizedImg.numpy().astype(int))
plt.show()
```

In [ ]:
```
np.expand_dims(resizedImg, 0)
```

In [311…
```
yhat = model.predict(np.expand_dims(resizedImg/255, 0))
print(yhat)

if yhat > 0.5:
    print("Prediction: Sad")
else:
    print("Prediction: Happy")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
[[0.9501678]]
Prediction: Sad
```

12. Saving the Model

In [ ]:
```
model.save(os.path.join('models','EmotionDetector.h5'))
```

In [314…
```
new_model = load_model(os.path.join('models', 'EmotionDetector.h5'))
yhatnew = new_model.predict(np.expand_dims(resizedImg/255, 0))
print(yhatnew)

if yhatnew > 0.5:
    print("Prediction: Sad")
```

```
else:
    print("Prediction: Happy")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be buil
t. `model.compile_metrics` will be empty until you train or evaluate the model.
WARNING:tensorflow:6 out of the last 13 calls to <function TensorFlowTrainer.make_pr
edict_function.<locals>.one_step_on_data_distributed at 0x7f21e9a02cb0> triggered t
f.function retracing. Tracing is expensive and the excessive number of tracings coul
d be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with
different shapes, (3) passing Python objects instead of tensors. For (1), please def
ine your @tf.function outside of the loop. For (2), @tf.function has reduce_retracin
g=True option that can avoid unnecessary retracing. For (3), please refer to http
s://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorfl
ow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 13 calls to <function TensorFlowTrainer.make_pr
edict_function.<locals>.one_step_on_data_distributed at 0x7f21e9a02cb0> triggered t
f.function retracing. Tracing is expensive and the excessive number of tracings coul
d be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with
different shapes, (3) passing Python objects instead of tensors. For (1), please def
ine your @tf.function outside of the loop. For (2), @tf.function has reduce_retracin
g=True option that can avoid unnecessary retracing. For (3), please refer to http
s://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorfl
ow.org/api_docs/python/tf/function for  more details.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
[[0.9501678]]
Prediction: Sad
```