Importing the libraries

```
import tensorflow as tf
print(tf.__version__)
```

```
2.18.0
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab.patches import cv2_imshow
import zipfile
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten, BatchNormalization
```

Loading the images

```
from google.colab import drive
drive.mount('/content/drive')
```

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

```
path = '/content/drive/MyDrive/Computer Vision Masterclass/Datasets/fer_images.zip'
zip_object = zipfile.ZipFile(file=path, mode='r')
zip_object.extractall('./')
zip_object.close()
```

```
tf.keras.preprocessing.image.load_img('/content/fer2013/train/Angry/1003.jpg')
```

```
image = tf.keras.preprocessing.image.load_img('/content/fer2013/train/Happy/1.jpg')
image
```

Train and test set

```
training_generator = ImageDataGenerator(rescale=1./255,
                                        rotation_range=7,
                                        horizontal_flip=True,
                                        zoom_range=0.2)
train_dataset = training_generator.flow_from_directory('/content/fer2013/train',
                                                       target_size = (48, 48),
                                                       batch_size = 16,
                                                       class_mode = 'categorical',
                                                       shuffle = True)
```

```
Found 28709 images belonging to 7 classes.
```

```
train_dataset.classes
```

```
array([0, 0, 0, ..., 6, 6, 6], dtype=int32)
```

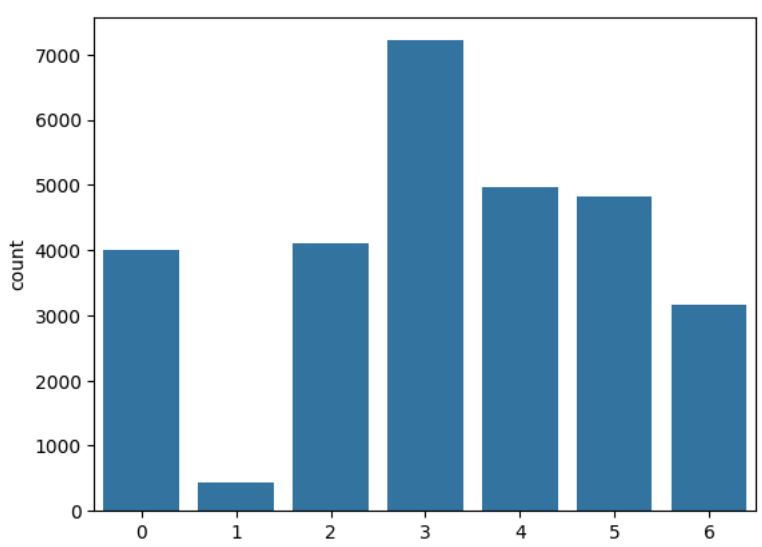
```
np.unique(train_dataset.classes, return_counts=True)
```

```
(array([0, 1, 2, 3, 4, 5, 6], dtype=int32),
 array([3995,  436, 4097, 7215, 4965, 4830, 3171]))
```

```
train_dataset.class_indices
```

```
{'Angry': 0,
 'Disgust': 1,
 'Fear': 2,
 'Happy': 3,
 'Neutral': 4,
 'Sad': 5,
 'Surprise': 6}
```

```
sns.countplot(x = train_dataset.classes);
```




```
test_generator = ImageDataGenerator(rescale=1./255)
test_dataset = test_generator.flow_from_directory('/content/fer2013/validation',
                                                  target_size = (48, 48),
                                                  batch_size = 1,
                                                  class_mode = 'categorical',
                                                  shuffle = False)
```

```
Found 3589 images belonging to 7 classes.
```

Building and training the convolutional neural network

```
2*2*2*32
```

```
256
```

```
num_detectors = 32
num_classes = 7
width, height = 48, 48
epochs = 70

network = Sequential() # i gonna define a sequence of layers

network.add(Conv2D(num_detectors, (3,3), activation='relu', padding = 'same', input_shape = (width, height, 3)))
network.add(BatchNormalization())
network.add(Conv2D(num_detectors, (3,3), activation='relu', padding = 'same'))
network.add(BatchNormalization())
network.add(MaxPooling2D(pool_size=(2,2)))
network.add(Dropout(0.2))

network.add(Conv2D(2*num_detectors, (3,3), activation='relu', padding = 'same'))
network.add(BatchNormalization())
network.add(Conv2D(2*num_detectors, (3,3), activation='relu', padding = 'same'))
network.add(BatchNormalization())
network.add(MaxPooling2D(pool_size=(2,2)))
network.add(Dropout(0.2))

network.add(Conv2D(2*2*num_detectors, (3,3), activation='relu', padding = 'same'))
network.add(BatchNormalization())
network.add(Conv2D(2*2*num_detectors, (3,3), activation='relu', padding = 'same'))
network.add(BatchNormalization())
network.add(MaxPooling2D(pool_size=(2,2)))
network.add(Dropout(0.2))

network.add(Conv2D(2*2*2*num_detectors, (3,3), activation='relu', padding = 'same'))
network.add(BatchNormalization())
network.add(Conv2D(2*2*2*num_detectors, (3,3), activation='relu', padding = 'same'))
```

```
network.add(Conv2D(2 * 2 * num_detectors, (3,3), activation='relu', padding = 'same'))
network.add(BatchNormalization())
network.add(MaxPooling2D(pool_size=(2,2)))
network.add(Dropout(0.2))

network.add(Flatten())

network.add(Dense(2 * num_detectors, activation='relu'))
network.add(BatchNormalization())
network.add(Dropout(0.2))

network.add(Dense(2 * num_detectors, activation='relu'))
network.add(BatchNormalization())
network.add(Dropout(0.2))

network.add(Dense(num_classes, activation='softmax'))
print(network.summary())
```

```
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 48, 48, 32)	896
batch_normalization (BatchNormalization)	(None, 48, 48, 32)	128
conv2d_1 (Conv2D)	(None, 48, 48, 32)	9,248
batch_normalization_1 (BatchNormalization)	(None, 48, 48, 32)	128
max_pooling2d (MaxPooling2D)	(None, 24, 24, 32)	0
dropout (Dropout)	(None, 24, 24, 32)	0
conv2d_2 (Conv2D)	(None, 24, 24, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 24, 24, 64)	256
conv2d_3 (Conv2D)	(None, 24, 24, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 24, 24, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 64)	0
dropout_1 (Dropout)	(None, 12, 12, 64)	0
conv2d_4 (Conv2D)	(None, 12, 12, 128)	73,856
batch_normalization_4 (BatchNormalization)	(None, 12, 12, 128)	512
conv2d_5 (Conv2D)	(None, 12, 12, 128)	147,584
batch_normalization_5 (BatchNormalization)	(None, 12, 12, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 128)	0
dropout_2 (Dropout)	(None, 6, 6, 128)	0
conv2d_6 (Conv2D)	(None, 6, 6, 256)	295,168
batch_normalization_6 (BatchNormalization)	(None, 6, 6, 256)	1,024
conv2d_7 (Conv2D)	(None, 6, 6, 256)	590,080
batch_normalization_7 (BatchNormalization)	(None, 6, 6, 256)	1,024
max_pooling2d_3 (MaxPooling2D)	(None, 3, 3, 256)	0
dropout_3 (Dropout)	(None, 3, 3, 256)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 64)	147,520
batch_normalization_8 (BatchNormalization)	(None, 64)	256
dropout_4 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 64)	4,160
batch_normalization_9 (BatchNormalization)	(None, 64)	256
dropout_5 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 7)	455

Total params: 1,328,743 (5.07 MB)
Trainable params: 1,326,567 (5.06 MB)
Non-trainable params: 2,176 (8.50 KB)

None

```
network.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
network.fit(train_dataset, epochs=epochs)
```

```
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 45s 25ms/step - accuracy: 0.6994 - loss: 0.8259
Epoch 39/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 44s 24ms/step - accuracy: 0.7000 - loss: 0.8263
Epoch 40/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 81s 24ms/step - accuracy: 0.7054 - loss: 0.8183
Epoch 41/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 42s 24ms/step - accuracy: 0.6944 - loss: 0.8335
Epoch 42/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 82s 24ms/step - accuracy: 0.7022 - loss: 0.8177
Epoch 43/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 45s 25ms/step - accuracy: 0.7064 - loss: 0.8067
Epoch 44/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 45s 25ms/step - accuracy: 0.7161 - loss: 0.7840
Epoch 45/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 43s 24ms/step - accuracy: 0.7147 - loss: 0.7920
Epoch 46/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 83s 24ms/step - accuracy: 0.7182 - loss: 0.7696
Epoch 47/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 44s 24ms/step - accuracy: 0.7184 - loss: 0.7840
Epoch 48/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 43s 24ms/step - accuracy: 0.7013 - loss: 0.8096
Epoch 49/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 82s 24ms/step - accuracy: 0.7169 - loss: 0.7778
Epoch 50/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 43s 24ms/step - accuracy: 0.7166 - loss: 0.7842
Epoch 51/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 43s 24ms/step - accuracy: 0.7178 - loss: 0.7736
Epoch 52/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 44s 24ms/step - accuracy: 0.7313 - loss: 0.7476
Epoch 53/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 43s 24ms/step - accuracy: 0.7231 - loss: 0.7654
Epoch 54/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 43s 24ms/step - accuracy: 0.7249 - loss: 0.7600
Epoch 55/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 43s 24ms/step - accuracy: 0.7287 - loss: 0.7474
Epoch 56/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 82s 24ms/step - accuracy: 0.7259 - loss: 0.7492
Epoch 57/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 42s 24ms/step - accuracy: 0.7310 - loss: 0.7455
Epoch 58/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 43s 24ms/step - accuracy: 0.7254 - loss: 0.7504
Epoch 59/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 43s 24ms/step - accuracy: 0.7328 - loss: 0.7368
Epoch 60/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 43s 24ms/step - accuracy: 0.7360 - loss: 0.7258
Epoch 61/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 83s 25ms/step - accuracy: 0.7381 - loss: 0.7175
Epoch 62/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 42s 24ms/step - accuracy: 0.7404 - loss: 0.7111
Epoch 63/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 43s 24ms/step - accuracy: 0.7421 - loss: 0.7054
Epoch 64/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 43s 24ms/step - accuracy: 0.7467 - loss: 0.7088
Epoch 65/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 45s 25ms/step - accuracy: 0.7500 - loss: 0.6920
Epoch 66/70
1795/1795 ━━━━━━━━━━━━━━━━━━━━ 81s 24ms/step - accuracy: 0.7515 - loss: 0.6956
Epoch 67/70
```

Saving and loading the model

```
model_json = network.to_json()
with open('network_emotions.json','w') as json_file:
  json_file.write(model_json)


from keras.models import save_model
network_saved = save_model(network, '/content/weights_emotions.hdf5')
```

```
idered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(m
```

```
from tensorflow.keras.models import model_from_json, Sequential

# Load the model architecture from the JSON file
with open('/content/network_emotions.json', 'r') as json_file:
    json_saved_model = json_file.read()

network_loaded = model_from_json(
    json_saved_model,
    custom_objects={'Sequential': Sequential}
)

# Load the model weights from the HDF5 file
```

```
network_loaded.load_weights('/content/drive/MyDrive/Computer Vision Masterclass/Weights/weights_emotions.hdf5')

network_loaded.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])

network_loaded.summary()
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 48, 48, 32)	896
batch_normalization (BatchNormalization)	(None, 48, 48, 32)	128
conv2d_1 (Conv2D)	(None, 48, 48, 32)	9,248
batch_normalization_1 (BatchNormalization)	(None, 48, 48, 32)	128
max_pooling2d (MaxPooling2D)	(None, 24, 24, 32)	0
dropout (Dropout)	(None, 24, 24, 32)	0
conv2d_2 (Conv2D)	(None, 24, 24, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 24, 24, 64)	256
conv2d_3 (Conv2D)	(None, 24, 24, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 24, 24, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 64)	0
dropout_1 (Dropout)	(None, 12, 12, 64)	0
conv2d_4 (Conv2D)	(None, 12, 12, 128)	73,856
batch_normalization_4 (BatchNormalization)	(None, 12, 12, 128)	512
conv2d_5 (Conv2D)	(None, 12, 12, 128)	147,584
batch_normalization_5 (BatchNormalization)	(None, 12, 12, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 128)	0
dropout_2 (Dropout)	(None, 6, 6, 128)	0
conv2d_6 (Conv2D)	(None, 6, 6, 256)	295,168
batch_normalization_6 (BatchNormalization)	(None, 6, 6, 256)	1,024
conv2d_7 (Conv2D)	(None, 6, 6, 256)	590,080
batch_normalization_7 (BatchNormalization)	(None, 6, 6, 256)	1,024
max_pooling2d_3 (MaxPooling2D)	(None, 3, 3, 256)	0
dropout_3 (Dropout)	(None, 3, 3, 256)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 64)	147,520
batch_normalization_8 (BatchNormalization)	(None, 64)	256
dropout_4 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 64)	4,160
batch_normalization_9 (BatchNormalization)	(None, 64)	256
dropout_5 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 7)	455

Total params: 1,328,743 (5.07 MB)
Trainable params: 1,326,567 (5.06 MB)

Evaluating the neural network

```
network_loaded.evaluate(test_dataset)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` cl
  self._warn_if_super_not_called()
3589/3589 ━━━━━━━━━━━━━━━━━━━━ 17s 4ms/step - accuracy: 0.5804 - loss: 1.3395
[1.5148499011993408, 0.5778768658638]
```

```
predictions = network_loaded.predict(test_dataset)
predictions
```

```
3589/3589 ━━━━━━━━━━━━━━━━━━━━ 12s 3ms/step
array([[9.1399026e-01, 5.2431659e-03, 2.0064995e-02, ..., 7.5534321e-03,
        3.9156068e-02, 1.2308669e-02],
       [8.3489537e-01, 1.3114783e-04, 1.6049375e-01, ..., 8.8334654e-04,
        3.4225823e-03, 1.4467943e-07],
       [9.6874458e-01, 7.4295519e-04, 1.7095286e-02, ..., 1.2566427e-03,
        7.5712553e-03, 4.9269065e-04],
       ...,
       [1.7553559e-03, 4.7019232e-05, 4.1051388e-02, ..., 1.6656774e-03,
        2.9383365e-03, 9.5245790e-01],
       [2.9105307e-03, 4.3033357e-05, 6.9266900e-02, ..., 3.6131206e-01,
        3.1379773e-03, 5.3026170e-01],
       [2.6819391e-02, 2.2662596e-03, 1.1300567e-01, ..., 4.7751591e-03,
        7.1213404e-03, 8.4558338e-01]], dtype=float32)
```

```
predictions = np.argmax(predictions, axis = 1)
predictions
```

```
array([0, 0, 0, ..., 6, 6, 6])
```

```
test_dataset.classes
```

```
array([0, 0, 0, ..., 6, 6, 6], dtype=int32)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(test_dataset.classes, predictions)
```

```
0.5778768459180831
```

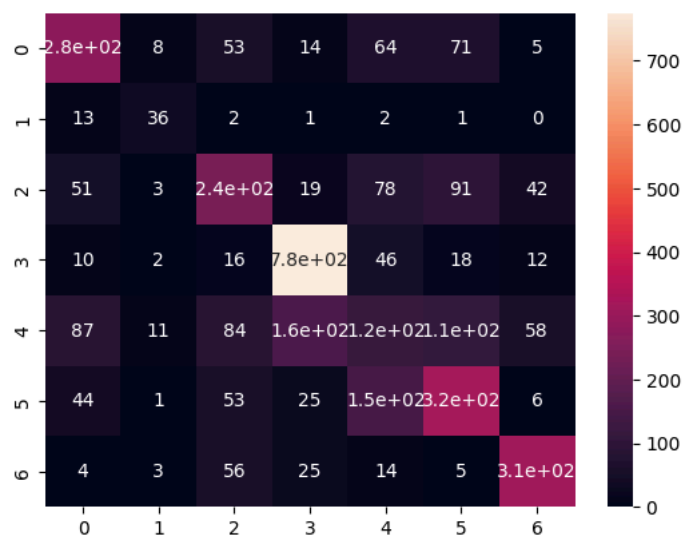
```
test_dataset.class_indices
```

```
{'Angry': 0,
 'Disgust': 1,
 'Fear': 2,
 'Happy': 3,
 'Neutral': 4,
 'Sad': 5,
 'Surprise': 6}
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(test_dataset.classes, predictions)
cm
```

```
array([[276,   8,  53,  14,  64,  71,   5],
       [ 13,  36,   2,   1,   2,   1,   0],
       [ 51,   3, 244,  19,  78,  91,  42],
       [ 10,   2,  16, 775,  46,  18,  12],
       [ 87,  11,  84, 160, 115, 111,  58],
       [ 44,   1,  53,  25, 146, 319,   6],
       [  4,   3,  56,  25,  14,   5, 309]])
```

```
sns.heatmap(cm, annot=True);
```




```
from sklearn.metrics import classification_report
print(classification_report(test_dataset.classes, predictions))
```

```
              precision    recall  f1-score   support

           0       0.57      0.56      0.57       491
           1       0.56      0.65      0.61        55
           2       0.48      0.46      0.47       528
           3       0.76      0.88      0.82       879
           4       0.25      0.18      0.21       626
           5       0.52      0.54      0.53       594
           6       0.72      0.74      0.73       416

    accuracy                           0.58      3589
   macro avg       0.55      0.57      0.56      3589
weighted avg       0.56      0.58      0.56      3589
```

Classifying one single image

```
image = cv2.imread('/content/IMG_20250506_214218.png')
cv2_imshow(image)
```

```
image.shape
```

```
(236, 225, 3)
```

```
face_detector = cv2.CascadeClassifier('/content/drive/MyDrive/Computer Vision Masterclass/Cascades/haarcascade_frontalface_default.xml')
```

```
original_image = image.copy()
faces = face_detector.detectMultiScale(original_image)
```

```
faces
```

```
array([[ 45,  38, 152, 152]], dtype=int32)
```

```
roi = image[10:200, 40:200]
cv2_imshow(roi)
```

```
roi.shape
```

```
(190, 160, 3)
```

```
roi = cv2.resize(roi, (48, 48))
cv2_imshow(roi)
```

```
roi.shape
```

```
(48, 48, 3)
```

```
roi
```

```
ndarray (48, 48, 3) show data
```

```
roi = roi / 255
roi
```

```
array([[[0.9372549 , 0.94509804, 0.9254902 ],
        [0.92941176, 0.94509804, 0.9254902 ],
        [0.9254902 , 0.94509804, 0.91764706],
        ...,
        [0.92156863, 0.9254902 , 0.90588235],
        [0.92156863, 0.9254902 , 0.90588235],
        [0.92156863, 0.9254902 , 0.90588235]],

       [[0.93333333, 0.94901961, 0.9254902 ],
        [0.92941176, 0.94509804, 0.92156863],
        [0.92156863, 0.9372549 , 0.91372549],
        ...,
        [0.92156863, 0.9254902 , 0.90588235],
        [0.92941176, 0.93333333, 0.91372549],
        [0.92156863, 0.9254902 , 0.90588235]],

       [[0.94117647, 0.95686275, 0.92941176],
        [0.93333333, 0.94901961, 0.9254902 ],
        [0.9254902 , 0.94117647, 0.91764706],
        ...,
        [0.91764706, 0.92156863, 0.90196078],
        [0.92941176, 0.92941176, 0.90980392],
        [0.92941176, 0.92941176, 0.90980392]],

       ...,

       [[0.80784314, 0.83137255, 0.82745098],
        [0.8       , 0.82352941, 0.81960784],
        [0.78823529, 0.81568627, 0.80784314],
        ...,
        [0.81960784, 0.81960784, 0.79215686],
        [0.82352941, 0.82352941, 0.79607843],
        [0.82745098, 0.82745098, 0.8       ]],

       [[0.76862745, 0.79607843, 0.78823529],
        [0.69803922, 0.74117647, 0.70980392],
        [0.52156863, 0.6       , 0.5372549 ],
        ...,
        [0.82745098, 0.81960784, 0.78431373],
        [0.82745098, 0.81568627, 0.79215686],
        [0.83529412, 0.82745098, 0.79607843]],

       [[0.55686275, 0.63529412, 0.57647059],
        [0.53333333, 0.61960784, 0.54901961],
        [0.54901961, 0.63921569, 0.56078431],
        ...,
        [0.81960784, 0.81568627, 0.77647059],
```

```
            [0.82745098, 0.82352941, 0.78431373],
            [0.83137255, 0.82352941, 0.78431373]]])
```

```
roi.shape
```

```
(48, 48, 3)
```

```
roi = np.expand_dims(roi, axis = 0)
roi.shape
```

```
(1, 48, 48, 3)
```

```
probs = network_loaded.predict(roi)
probs
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 954ms/step
array([[1.2675990e-04, 1.9988229e-06, 5.5030128e-04, 8.9367753e-01,
        1.0431930e-01, 1.1285383e-03, 1.9558656e-04]], dtype=float32)
```

```
result = np.argmax(probs)
result
```

```
np.int64(3)
```

```
test_dataset.class_indices
```

```
{'Angry': 0,
 'Disgust': 1,
 'Fear': 2,
 'Happy': 3,
 'Neutral': 4,
 'Sad': 5,
 'Surprise': 6}
```

classifying multiple images

```
image = cv2.imread('/content/drive/MyDrive/Computer Vision Masterclass/Images/faces_emotions.png')
cv2_imshow(image)
```

```
faces = face_detector.detectMultiScale(image)
faces
```

```
array([[625,  49,  91,  91],
       [224,  35,  90,  90],
       [ 23,  41,  92,  92],
       [420,  43,  97,  97],
       [420, 242,  97,  97],
       [ 18, 243,  98,  98],
       [229, 242,  85,  85],
       [627, 241,  91,  91]], dtype=int32)
```

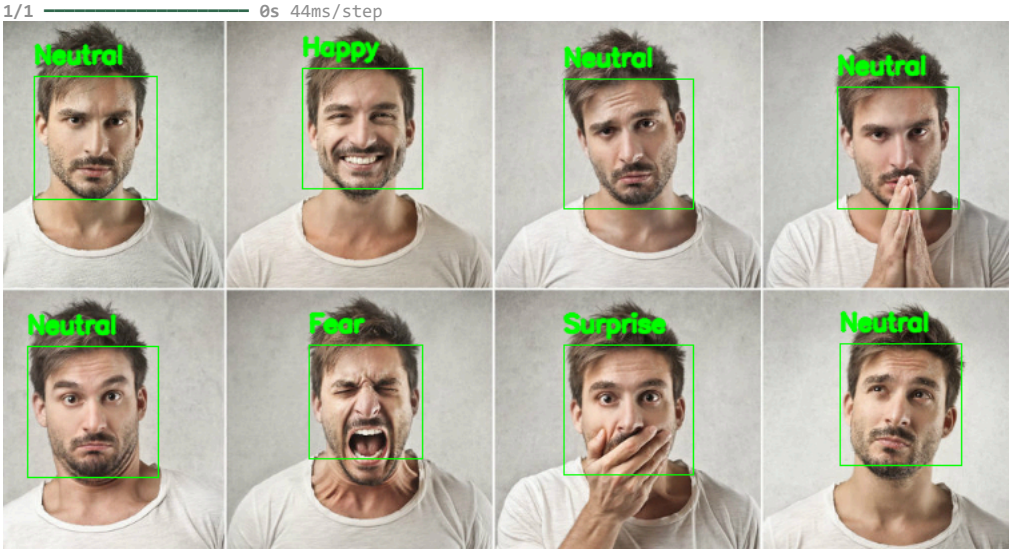
```
test_dataset.class_indices.keys()
```

```
dict_keys(['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise'])
```

```
emotions = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']


for (x, y, w, h) in faces:
  cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 1)
  roi = image[y:y + h, x:x + w]
  #cv2_imshow(roi)
  roi = cv2.resize(roi, (48, 48))
  #cv2_imshow(roi)
  roi = roi / 255
  roi = np.expand_dims(roi, axis = 0)
  #print(roi.shape)
  prediction = network_loaded.predict(roi)
  #print(prediction)
  cv2.putText(image, emotions[np.argmax(prediction)], (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0,255,0), 2, cv2.LINE_AA)
cv2_imshow(image)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
```



classifying video emotions

```
cap = cv2.VideoCapture('/content/drive/MyDrive/Computer Vision Masterclass/Videos/emotion_test01.mp4')
connected, video = cap.read()
print(connected, video.shape)
```

```
True (360, 640, 3)
```

```
# fourcc.org
save_path = '/content/drive/MyDrive/Computer Vision Masterclass/Videos/emotion_test01_result.avi'
fourcc = cv2.VideoWriter_fourcc(*'XVID')
fps = 24
output_video = cv2.VideoWriter(save_path, fourcc, fps, (video.shape[1], video.shape[0]))


while (cv2.waitKey(1) < 0):
  connected, frame = cap.read()
  if not connected:
    break
  faces = face_detector.detectMultiScale(frame, scaleFactor=1.2, minNeighbors=5, minSize=(30,30))
  if len(faces) > 0:
    for (x, y, w, h) in faces:
      frame = cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
      roi = frame[y:y + h, x:x + w]
      roi = cv2.resize(roi, (48, 48))
      roi = roi / 255
      roi = np.expand_dims(roi, axis = 0)
      prediction = network_loaded.predict(roi)

      if prediction is not None:
        result = np.argmax(prediction)
        cv2.putText(frame, emotions[result], (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,255,255), 1, cv2.LINE_AA)
```