```
from google.colab import files
uploaded = files.upload()
for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
name=fn, length=len(uploaded[fn])))
```

Choose Files archive (1).zip

- **archive (1).zip**(application/x-zip-compressed) - 63252113 bytes, last modified: 7/31/2023 - 100% done

```
Saving archive (1).zip to archive (1).zip
User uploaded file "archive (1).zip" with length 63252113 bytes
```

```
from zipfile import ZipFile
file_name = "archive (1).zip"
with ZipFile(file_name, 'r') as zip:
 zip.extractall()
 print("Done")
```

```
Done
```

```
import numpy as np
import cv2
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D
from keras.optimizers import Adam
from keras.layers import MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
```

```
train_dir = 'train'
val_dir = 'test'
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
train_dir,
target_size=(48,48),
batch_size=64,
color_mode="grayscale",
class_mode='categorical')
validation_generator = val_datagen.flow_from_directory(
val_dir,
target_size=(48,48),
batch_size=64,
color_mode="grayscale",
class_mode='categorical')
```

```
Found 28709 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

```
emotion_model = Sequential()
emotion_model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(48,48,1)))
emotion_model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))
emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))
emotion_model.add(Flatten())
emotion_model.add(Dense(1024, activation='relu'))
emotion_model.add(Dropout(0.5))
emotion_model.add(Dense(7, activation='softmax'))
emotion_model.compile(loss='categorical_crossentropy',optimizer=Adam(lr=0.0001, decay=1e-6),metrics=['accuracy'])
emotion_model_info = emotion_model.fit_generator(
train_generator,
steps_per_epoch=28709 // 64,
epochs=65,
validation_data=validation_generator,
validation_steps=7178 // 64)
```

```
448/448 [==============================] - 14s 31ms/step - loss: 0.5871 - accuracy: 0.7877 - val_loss: 1.1058 - val_accuracy: 0.
Epoch 38/65
448/448 [==============================] - 15s 32ms/step - loss: 0.5692 - accuracy: 0.7952 - val_loss: 1.1213 - val_accuracy: 0.
Epoch 39/65
448/448 [==============================] - 14s 32ms/step - loss: 0.5509 - accuracy: 0.8006 - val_loss: 1.1125 - val_accuracy: 0.
Epoch 40/65
448/448 [==============================] - 14s 31ms/step - loss: 0.5259 - accuracy: 0.8114 - val_loss: 1.1074 - val_accuracy: 0.
Epoch 41/65
448/448 [==============================] - 14s 32ms/step - loss: 0.5165 - accuracy: 0.8129 - val_loss: 1.1296 - val_accuracy: 0.
Epoch 42/65
448/448 [==============================] - 15s 32ms/step - loss: 0.4931 - accuracy: 0.8227 - val_loss: 1.1368 - val_accuracy: 0.
Epoch 43/65
448/448 [==============================] - 15s 33ms/step - loss: 0.4716 - accuracy: 0.8287 - val_loss: 1.1547 - val_accuracy: 0.
Epoch 44/65
448/448 [==============================] - 14s 31ms/step - loss: 0.4635 - accuracy: 0.8290 - val_loss: 1.1386 - val_accuracy: 0.
Epoch 45/65
448/448 [==============================] - 14s 32ms/step - loss: 0.4528 - accuracy: 0.8360 - val_loss: 1.1606 - val_accuracy: 0.
Epoch 46/65
448/448 [==============================] - 14s 32ms/step - loss: 0.4325 - accuracy: 0.8422 - val_loss: 1.1684 - val_accuracy: 0.
Epoch 47/65
448/448 [==============================] - 14s 32ms/step - loss: 0.4269 - accuracy: 0.8472 - val_loss: 1.1693 - val_accuracy: 0.
Epoch 48/65
448/448 [==============================] - 14s 31ms/step - loss: 0.4032 - accuracy: 0.8562 - val_loss: 1.1876 - val_accuracy: 0.
Epoch 49/65
448/448 [==============================] - 14s 32ms/step - loss: 0.3939 - accuracy: 0.8587 - val_loss: 1.1937 - val_accuracy: 0.
Epoch 50/65
448/448 [==============================] - 14s 31ms/step - loss: 0.3802 - accuracy: 0.8617 - val_loss: 1.2007 - val_accuracy: 0.
Epoch 51/65
448/448 [==============================] - 14s 31ms/step - loss: 0.3663 - accuracy: 0.8678 - val_loss: 1.2288 - val_accuracy: 0.
Epoch 52/65
448/448 [==============================] - 14s 32ms/step - loss: 0.3643 - accuracy: 0.8693 - val_loss: 1.2241 - val_accuracy: 0.
Epoch 53/65
448/448 [==============================] - 14s 32ms/step - loss: 0.3511 - accuracy: 0.8758 - val_loss: 1.2247 - val_accuracy: 0.
Epoch 54/65
448/448 [==============================] - 14s 32ms/step - loss: 0.3379 - accuracy: 0.8803 - val_loss: 1.2512 - val_accuracy: 0.
Epoch 55/65
448/448 [==============================] - 14s 31ms/step - loss: 0.3306 - accuracy: 0.8801 - val_loss: 1.2652 - val_accuracy: 0.
Epoch 56/65
448/448 [==============================] - 14s 32ms/step - loss: 0.3198 - accuracy: 0.8840 - val_loss: 1.2709 - val_accuracy: 0.
Epoch 57/65
448/448 [==============================] - 14s 32ms/step - loss: 0.3088 - accuracy: 0.8920 - val_loss: 1.2697 - val_accuracy: 0.
Epoch 58/65
448/448 [==============================] - 14s 32ms/step - loss: 0.3042 - accuracy: 0.8930 - val_loss: 1.2797 - val_accuracy: 0.
Epoch 59/65
448/448 [==============================] - 14s 31ms/step - loss: 0.2983 - accuracy: 0.8945 - val_loss: 1.2999 - val_accuracy: 0.
Epoch 60/65
448/448 [==============================] - 14s 31ms/step - loss: 0.2868 - accuracy: 0.8998 - val_loss: 1.2997 - val_accuracy: 0.
Epoch 61/65
448/448 [==============================] - 15s 33ms/step - loss: 0.2798 - accuracy: 0.8999 - val_loss: 1.3125 - val_accuracy: 0.
Epoch 62/65
448/448 [==============================] - 14s 31ms/step - loss: 0.2715 - accuracy: 0.9029 - val_loss: 1.2942 - val_accuracy: 0.
Epoch 63/65
448/448 [==============================] - 15s 32ms/step - loss: 0.2640 - accuracy: 0.9047 - val_loss: 1.3145 - val_accuracy: 0.
Epoch 64/65
448/448 [==============================] - 15s 33ms/step - loss: 0.2624 - accuracy: 0.9078 - val_loss: 1.3271 - val_accuracy: 0.
Epoch 65/65
448/448 [==============================] - 14s 31ms/step - loss: 0.2484 - accuracy: 0.9122 - val_loss: 1.3450 - val_accuracy: 0.
```

```
emotion_model.save('model.h5')
```

```
from keras.models import load_model
emotion_model = load_model('model.h5')
```

```
def emotion_analysis(emotions):
  objects = ('angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral')
  y_pos = np.arange(len(objects))
  plt.bar(y_pos, emotions, align='center', alpha=0.5)
  plt.xticks(y_pos, objects)
  plt.ylabel('percentage')
  plt.title('emotion')
  plt.show()
```

```
import matplotlib.pyplot as plt
```

```
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode
def take_photo(filename='photo.jpg', quality=0.8):
  js = Javascript('''
   async function takePhoto(quality) {
    const div = document.createElement('div');
```

```
    const capture = document.createElement('button');
    capture.textContent = 'Capture';
    div.appendChild(capture);
    const video = document.createElement('video');
    video.style.display = 'block';
    const stream = await navigator.mediaDevices.getUserMedia({video: true});
    document.body.appendChild(div);
    div.appendChild(video);
    video.srcObject = stream;
    await video.play();
    // Resize the output to fit the video element.
    google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);
    // Wait for Capture to be clicked.
    await new Promise((resolve) => capture.onclick = resolve);
    const canvas = document.createElement('canvas');
    canvas.width = video.videoWidth;
    canvas.height = video.videoHeight;
    canvas.getContext('2d').drawImage(video, 0, 0);
    stream.getVideoTracks()[0].stop();
    div.remove();
    return canvas.toDataURL('image/jpeg', quality);
}
''')
  display(js)
  data = eval_js('takePhoto({})'.format(quality))
  binary = b64decode(data.split(',')[1])
  with open(filename, 'wb') as f:
    f.write(binary)
  return filename



take_photo()
```

```
    'photo.jpg'
```

```
from tensorflow.keras.utils import load_img
from keras.models import load_model
import matplotlib.pyplot as plt
import cv2
from tensorflow.keras.utils import load_img
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import img_to_array
import numpy as np
import matplotlib.pyplot as plt
file = '/content/angry.jpeg'
true_image =load_img(file)
img =load_img(file, color_mode="grayscale", target_size=(48, 48))
x =img_to_array(img)
x = np.expand_dims(x, axis = 0)
x /= 255
custom = emotion_model.predict(x)
emotion_analysis(custom[0])
x = np.array(x, 'float32')
x = x.reshape([48, 48]);
plt.imshow(true_image)
plt.show()
```

```
1/1 [==============================] - 0s 78ms/step
```