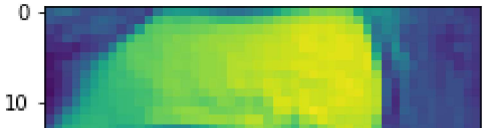
```
import keras
from keras.models import Sequential
from keras.layers import Dense,Flatten,Dropout
from keras.layers.convolutional import Conv2D,MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator ,load_img
#from keras.layers.normalization import BatchNormalization


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
from sklearn.model_selection import train_test_split


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

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

```
#Importing data from csv file
data=pd.read_csv("/content/drive/MyDrive/fer2013.csv")


labels=data.iloc[:,[0]].values


pixels=data['pixels']


#Facial Expressions
Expressions={0:"Angry",1:"Disgust",2:"Fear",3:"Happy",4:"Sad",5:"Surprise",6:"Neutral"}
from tensorflow.keras.utils import to_categorical
labels = to_categorical(labels,len(Expressions))


#converting pixels to Gray Scale images of 48x48
images = np.array([np.fromstring(pixel, dtype=int, sep=" ")for pixel in pixels])
images=images/255.0
images = images.reshape(images.shape[0],48,48,1).astype('float32')


plt.imshow(images[0][:,:,0])
Expressions[labels[0][0]]
```

```
'Disgust'
```



```
train_images,test_images,train_labels,test_labels = train_test_split(images,labels,test_si
```

```
train_labels
```

```
array([[0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]], dtype=float32)
```

```
from tensorflow.keras.layers import BatchNormalization
def create_convolutional_model(classes):
    model = Sequential()
    model.add(Conv2D(32,kernel_size=(2,2),strides=(1,1),activation='relu',input_shape=(48,
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(filters=64,kernel_size=(2,2),strides=(1,1),activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2),strides=(1,1)))
    model.add(Dropout(0.25))#to prevent neural network from overfitting

    model.add(Conv2D(filters=128,kernel_size=(2,2),strides=(1,1),activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2),strides=(1,1)))
    model.add(Dropout(0.25))

    model.add(Conv2D(filters=256,kernel_size=(2,2),strides=(1,1),activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2),strides=(1,1)))
    model.add(Dropout(0.25))

    model.add(Flatten())

    model.add(Dense(256,activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

    model.add(Dense(512,activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

    model.add(Dense(classes,activation='softmax'))
```

```
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

    return model


classes=7
model = create_convolutional_model(classes)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 47, 47, 32)        160

batch_normalization (BatchN  (None, 47, 47, 32)       128
ormalization)

max_pooling2d (MaxPooling2D  (None, 23, 23, 32)       0
)

dropout (Dropout)            (None, 23, 23, 32)        0

conv2d_1 (Conv2D)            (None, 22, 22, 64)        8256

batch_normalization_1 (Batc  (None, 22, 22, 64)       256
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 21, 21, 64)       0
2D)

dropout_1 (Dropout)          (None, 21, 21, 64)        0

conv2d_2 (Conv2D)            (None, 20, 20, 128)       32896

batch_normalization_2 (Batc  (None, 20, 20, 128)      512
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 19, 19, 128)      0
2D)

dropout_2 (Dropout)          (None, 19, 19, 128)       0

conv2d_3 (Conv2D)            (None, 18, 18, 256)       131328

batch_normalization_3 (Batc  (None, 18, 18, 256)      1024
hNormalization)

max_pooling2d_3 (MaxPooling  (None, 17, 17, 256)      0
2D)

dropout_3 (Dropout)          (None, 17, 17, 256)       0

flatten (Flatten)            (None, 73984)             0

dense (Dense)                (None, 256)               18940160

batch_normalization_4 (Batc  (None, 256)              1024
hNormalization)
```

```
dropout_4 (Dropout)         (None, 256)               0

dense_1 (Dense)             (None, 512)               131584

batch_normalization_5 (Batc  (None, 512)              2048
hNormalization)
```

```
model.fit(train_images,train_labels,batch_size=105,epochs=30,verbose=2)
```

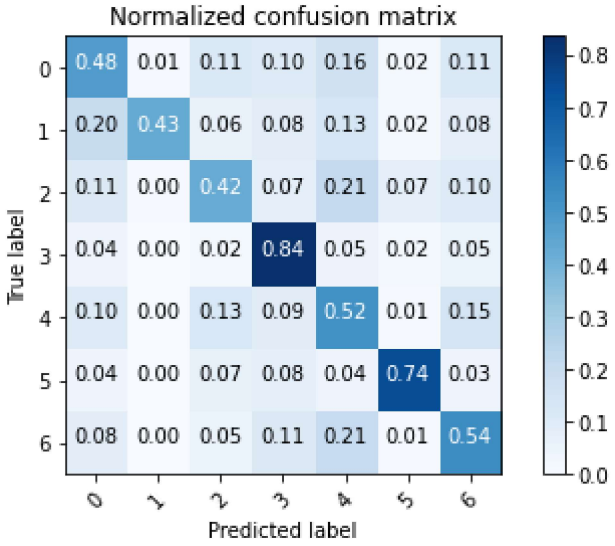
```
Epoch 1/30
274/274 - 21s - loss: 1.8808 - accuracy: 0.3236 - 21s/epoch - 77ms/step
Epoch 2/30
274/274 - 9s - loss: 1.4958 - accuracy: 0.4365 - 9s/epoch - 35ms/step
Epoch 3/30
274/274 - 10s - loss: 1.3579 - accuracy: 0.4860 - 10s/epoch - 35ms/step
Epoch 4/30
274/274 - 10s - loss: 1.2930 - accuracy: 0.5059 - 10s/epoch - 35ms/step
Epoch 5/30
274/274 - 10s - loss: 1.1818 - accuracy: 0.5534 - 10s/epoch - 35ms/step
Epoch 6/30
274/274 - 10s - loss: 1.0959 - accuracy: 0.5859 - 10s/epoch - 36ms/step
Epoch 7/30
274/274 - 10s - loss: 1.0310 - accuracy: 0.6097 - 10s/epoch - 36ms/step
Epoch 8/30
274/274 - 10s - loss: 0.9600 - accuracy: 0.6389 - 10s/epoch - 36ms/step
Epoch 9/30
274/274 - 10s - loss: 0.8970 - accuracy: 0.6637 - 10s/epoch - 37ms/step
Epoch 10/30
274/274 - 10s - loss: 0.8119 - accuracy: 0.6980 - 10s/epoch - 37ms/step
Epoch 11/30
274/274 - 10s - loss: 0.7393 - accuracy: 0.7244 - 10s/epoch - 37ms/step
Epoch 12/30
274/274 - 10s - loss: 0.6405 - accuracy: 0.7623 - 10s/epoch - 37ms/step
Epoch 13/30
274/274 - 10s - loss: 0.5639 - accuracy: 0.7966 - 10s/epoch - 37ms/step
Epoch 14/30
274/274 - 10s - loss: 0.4916 - accuracy: 0.8202 - 10s/epoch - 36ms/step
Epoch 15/30
274/274 - 10s - loss: 0.4183 - accuracy: 0.8486 - 10s/epoch - 36ms/step
Epoch 16/30
274/274 - 10s - loss: 0.3679 - accuracy: 0.8692 - 10s/epoch - 36ms/step
Epoch 17/30
274/274 - 10s - loss: 0.3258 - accuracy: 0.8833 - 10s/epoch - 36ms/step
Epoch 18/30
274/274 - 10s - loss: 0.3100 - accuracy: 0.8887 - 10s/epoch - 37ms/step
Epoch 19/30
274/274 - 10s - loss: 0.2704 - accuracy: 0.9056 - 10s/epoch - 37ms/step
Epoch 20/30
274/274 - 10s - loss: 0.2342 - accuracy: 0.9169 - 10s/epoch - 37ms/step
Epoch 21/30
274/274 - 10s - loss: 0.2268 - accuracy: 0.9212 - 10s/epoch - 37ms/step
Epoch 22/30
274/274 - 10s - loss: 0.2114 - accuracy: 0.9249 - 10s/epoch - 37ms/step
Epoch 23/30
274/274 - 10s - loss: 0.1883 - accuracy: 0.9326 - 10s/epoch - 37ms/step
Epoch 24/30
274/274 - 10s - loss: 0.1801 - accuracy: 0.9367 - 10s/epoch - 37ms/step
Epoch 25/30
```

```
274/274 - 10s - loss: 0.1728 - accuracy: 0.9388 - 10s/epoch - 37ms/step
Epoch 26/30
274/274 - 10s - loss: 0.1725 - accuracy: 0.9386 - 10s/epoch - 37ms/step
Epoch 27/30
274/274 - 10s - loss: 0.1527 - accuracy: 0.9492 - 10s/epoch - 37ms/step
Epoch 28/30
274/274 - 10s - loss: 0.1575 - accuracy: 0.9450 - 10s/epoch - 37ms/step
Epoch 29/30
274/274 - 10s - loss: 0.1396 - accuracy: 0.9532 - 10s/epoch - 37ms/step
```

```
label_pred=model.predict(test_images)
label_pred=np.argmax(label_pred,axis = 1)


import itertools
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

test_labels=np.argmax(test_labels,axis=1)
# Compute confusion matrix
cnf_matrix = confusion_matrix(test_labels,label_pred)
class_names=Expressions
```

```
# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Normalized confusion matrix
[[4.82830385e-01 8.32466181e-03 1.13423517e-01 1.03017690e-01
  1.60249740e-01 1.76899063e-02 1.14464100e-01]
 [1.98113208e-01 4.33962264e-01 5.66037736e-02 8.49056604e-02
  1.32075472e-01 1.88679245e-02 7.54716981e-02]
 [1.10337972e-01 4.97017893e-03 4.24453280e-01 7.35586481e-02
  2.09741551e-01 7.45526839e-02 1.02385686e-01]
 [3.53837779e-02 0.00000000e+00 1.52422428e-02 8.35601524e-01
  4.57267284e-02 1.52422428e-02 5.28034839e-02]
 [1.01506741e-01 0.00000000e+00 1.29262490e-01 9.27835052e-02
  5.20222046e-01 9.51625694e-03 1.46708961e-01]
 [3.62953692e-02 1.25156446e-03 7.13391740e-02 7.75969962e-02
  4.00500626e-02 7.39674593e-01 3.37922403e-02]
 [8.36092715e-02 8.27814570e-04 5.21523179e-02 1.10099338e-01
  2.07781457e-01 9.10596026e-03 5.36423841e-01]]
```




```
filename='model_weights.hdf5'
model.save_weights(filename,overwrite=True)
```

```
filename='model_weights.hdf5'
model.load_weights(filename)
```

```
import cv2
def make_prediction(unknown):
    unknown=cv2.resize(unknown,(48,48))
    unknown=unknown/255.0
    unknown=np.array(unknown).reshape(-1,48,48,1)
    predict=np.argmax(model.predict(unknown),axis = 1)
    return predict[0]
```

```
from google.colab.patches import cv2_imshow
```

```
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


from IPython.display import Image
try:
  filename = take_photo()
  print('Saved to {}'.format(filename))

  # Show the image which was just taken.
  display(Image(filename))
except Exception as err:
  # Errors will be thrown if the user does not have a webcam or if they do not
  # grant the page permission to access it.
  print(str(err))
```

```
Saved to photo.jpg
```

```
import cv2 as cv
def face_in_video():
    face_cascade = cv.CascadeClassifier("/content/drive/MyDrive/haarcascade_frontalface_de
    while True:
        !curl -o photo.jpg /content/photo.jpg
        import cv2
        img = cv2.imread('photo.jpg', cv2.IMREAD_UNCHANGED)
        gray = cv.cvtColor(img,cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(img)

        for (x,y,w,h) in faces:
            sub_face = gray[y:y+h, x:x+w]
            cv.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
            res=make_prediction(sub_face)
            font = cv.FONT_HERSHEY_SIMPLEX
            cv.putText(img,str(Expressions[res]),(x,y-5),font,0.5,(205,200,50),1,cv.LINE_A
        cv2_imshow(img)
        break
    cv.destroyAllWindows()
```

```
face_in_video()
```

```
curl: (3) <url> malformed
```