Face Emotion Recognition

Project idea

- We used a data set containing 700 images from 7 different classes that indicate facial expressions

- The categories in the data set are:

neutral

happy

surprise

angry

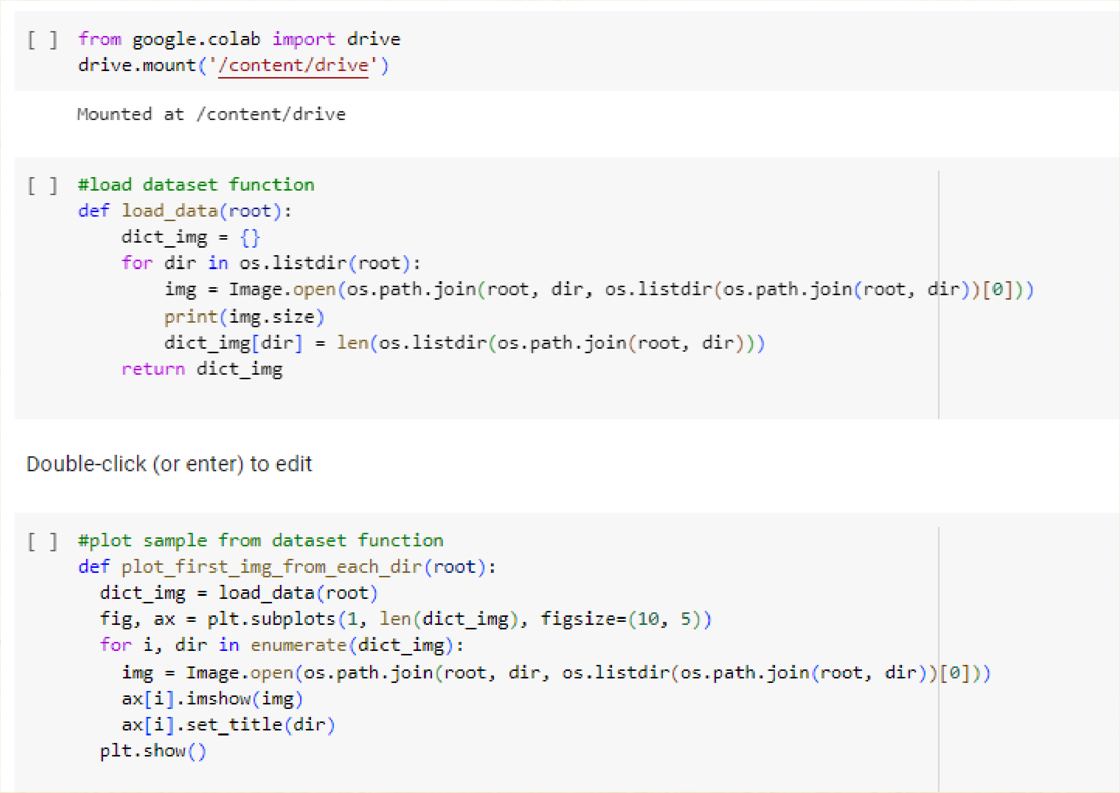
sad

fear

disgust

Dataset

- First, we downloaded the dataset to our drive and uploaded it to Collab
- Then load the dataset and display an image from each category using functions



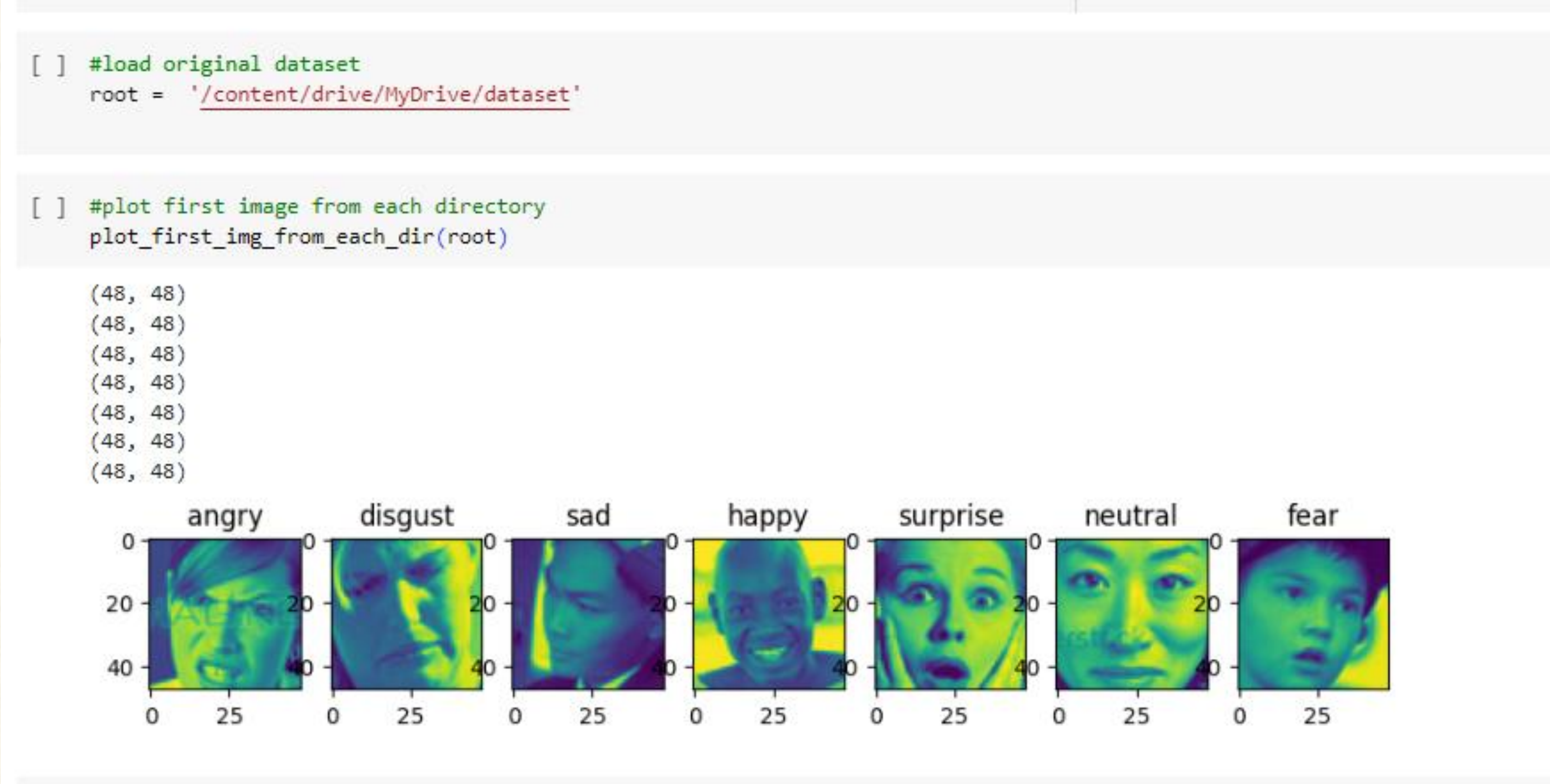
```
[ ]  from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive
```

```
[ ]  #load dataset function
     def load_data(root):
         dict_img = {}
         for dir in os.listdir(root):
             img = Image.open(os.path.join(root, dir, os.listdir(os.path.join(root, dir))[0]))
             print(img.size)
             dict_img[dir] = len(os.listdir(os.path.join(root, dir)))
         return dict_img
```

Double-click (or enter) to edit

```
[ ]  #plot sample from dataset function
     def plot_first_img_from_each_dir(root):
       dict_img = load_data(root)
       fig, ax = plt.subplots(1, len(dict_img), figsize=(10, 5))
       for i, dir in enumerate(dict_img):
         img = Image.open(os.path.join(root, dir, os.listdir(os.path.join(root, dir))[0]))
         ax[i].imshow(img)
         ax[i].set_title(dir)
       plt.show()
```

- Original dataset



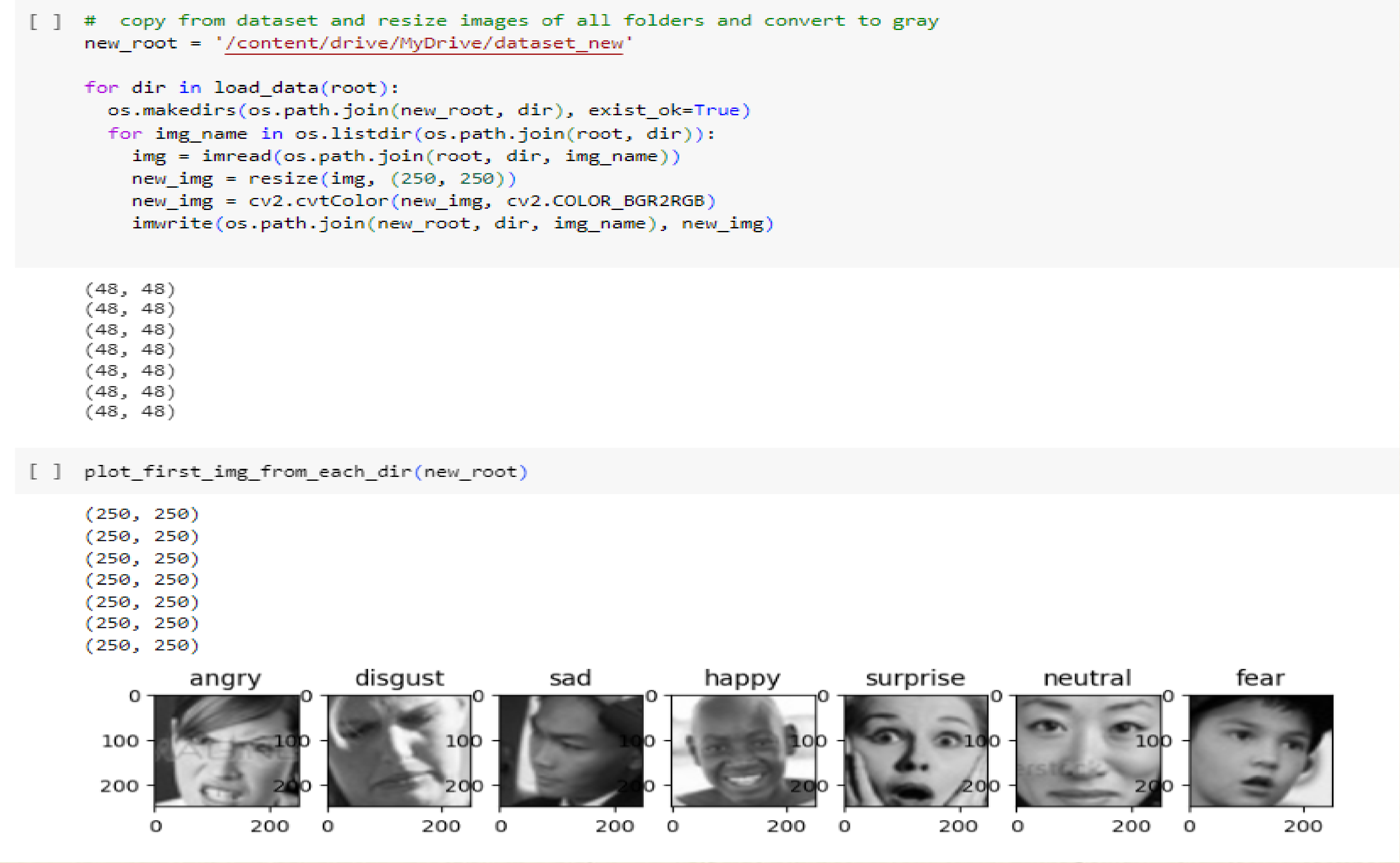
```
[ ]  #load original dataset
     root =  '/content/drive/MyDrive/dataset'
```

```
[ ]  #plot first image from each directory
     plot_first_img_from_each_dir(root)
```

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

Preprocessing

- Convert images to RGB
- Resize images from 48*48 to 250*250

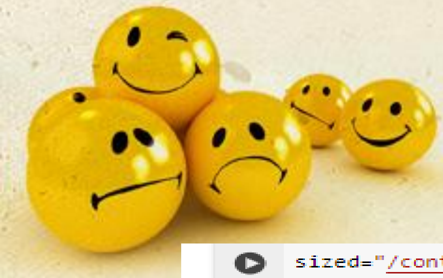


```
[ ]  #  copy from dataset and resize images of all folders and convert to gray
     new_root = '/content/drive/MyDrive/dataset_new'

     for dir in load_data(root):
       os.makedirs(os.path.join(new_root, dir), exist_ok=True)
       for img_name in os.listdir(os.path.join(root, dir)):
         img = imread(os.path.join(root, dir, img_name))
         new_img = resize(img, (250, 250))
         new_img = cv2.cvtColor(new_img, cv2.COLOR_BGR2RGB)
         imwrite(os.path.join(new_root, dir, img_name), new_img)
```

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

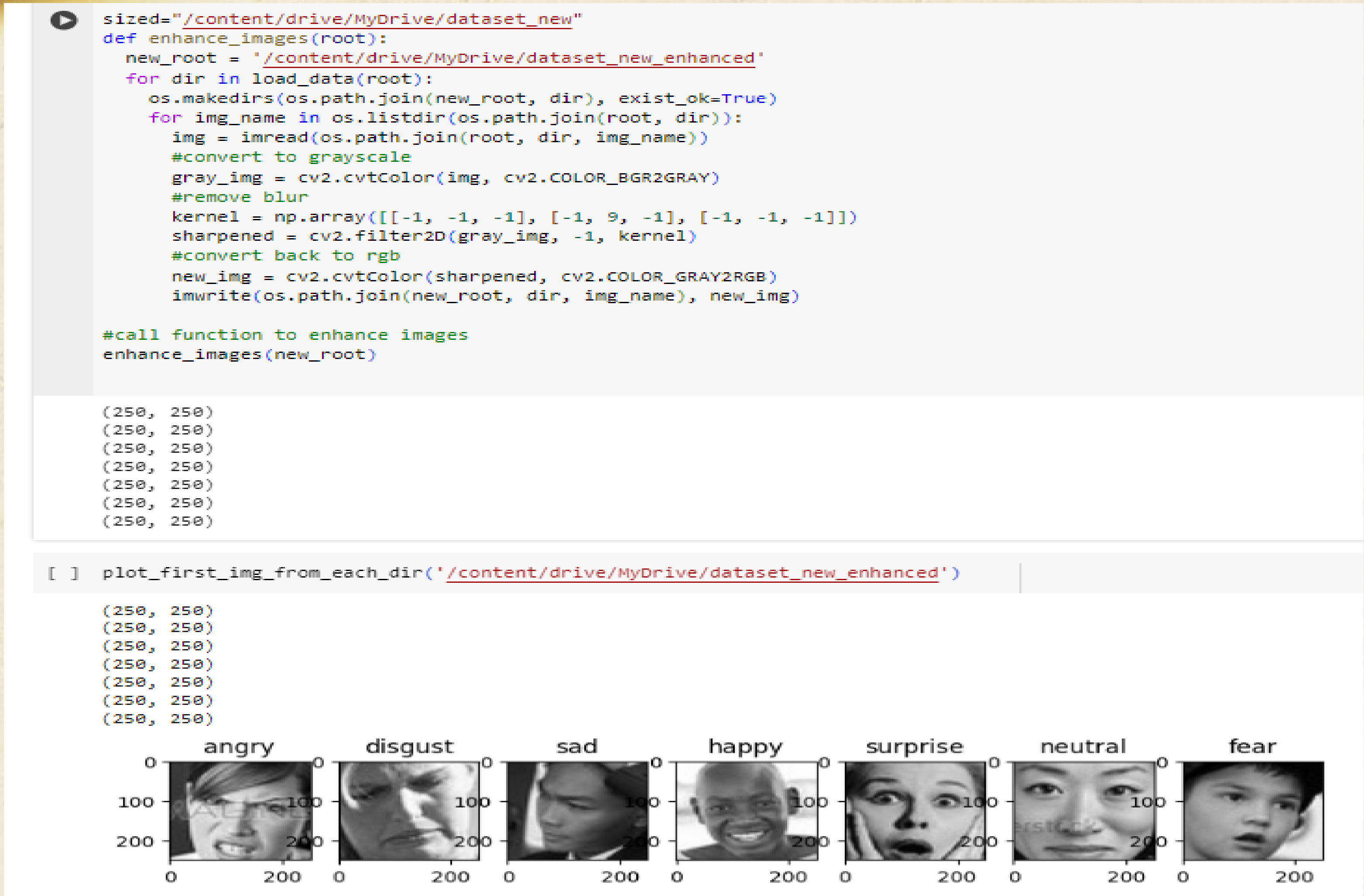
```
[ ]  plot_first_img_from_each_dir(new_root)
```

```
(250, 250)
(250, 250)
(250, 250)
(250, 250)
(250, 250)
(250, 250)
(250, 250)
```



Preprocessing

- Convert images to gray
- Enhancement images by filter to **remove the blur**



```
sized="/content/drive/MyDrive/dataset_new"
def enhance_images(root):
  new_root = '/content/drive/MyDrive/dataset_new_enhanced'
  for dir in load_data(root):
    os.makedirs(os.path.join(new_root, dir), exist_ok=True)
    for img_name in os.listdir(os.path.join(root, dir)):
      img = imread(os.path.join(root, dir, img_name))
      #convert to grayscale
      gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
      #remove blur
      kernel = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
      sharpened = cv2.filter2D(gray_img, -1, kernel)
      #convert back to rgb
      new_img = cv2.cvtColor(sharpened, cv2.COLOR_GRAY2RGB)
      imwrite(os.path.join(new_root, dir, img_name), new_img)

#call function to enhance images
enhance_images(new_root)
```

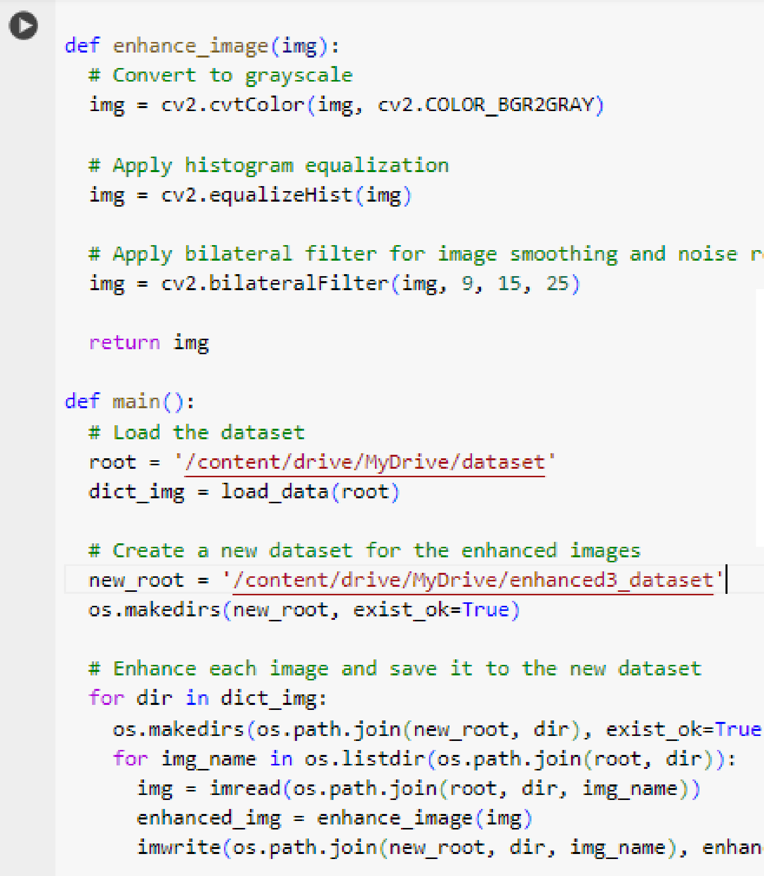
```
(250, 250)
(250, 250)
(250, 250)
(250, 250)
(250, 250)
(250, 250)
(250, 250)
```

```
[ ]  plot_first_img_from_each_dir('/content/drive/MyDrive/dataset_new_enhanced')
```

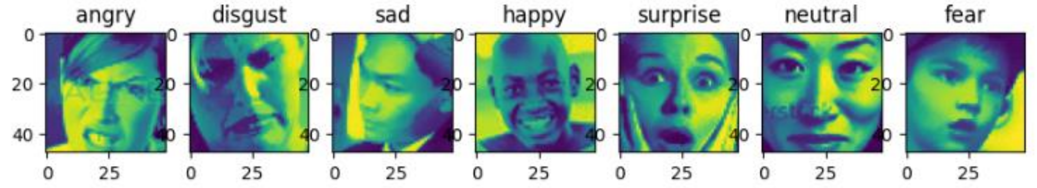
```
(250, 250)
(250, 250)
(250, 250)
(250, 250)
(250, 250)
(250, 250)
(250, 250)
```

Preprocessing

- We tried using another algorithm in the preprocessing stage by applying **histogram equalization** and **bilateral filter** for image smoothing and noise reduction.

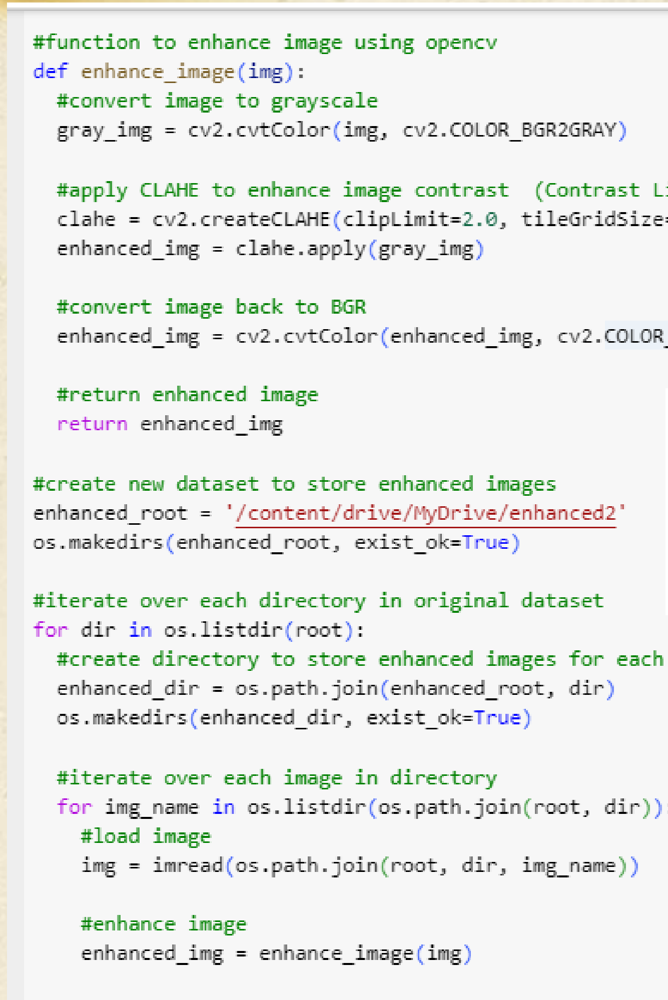


```
def enhance_image(img):
  # Convert to grayscale
  img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

  # Apply histogram equalization
  img = cv2.equalizeHist(img)

  # Apply bilateral filter for image smoothing and noise reduction
  img = cv2.bilateralFilter(img, 9, 15, 25)

  return img

def main():
  # Load the dataset
  root = '/content/drive/MyDrive/dataset'
  dict_img = load_data(root)

  # Create a new dataset for the enhanced images
  new_root = '/content/drive/MyDrive/enhanced3_dataset'
  os.makedirs(new_root, exist_ok=True)

  # Enhance each image and save it to the new dataset
  for dir in dict_img:
    os.makedirs(os.path.join(new_root, dir), exist_ok=True)
    for img_name in os.listdir(os.path.join(root, dir)):
      img = imread(os.path.join(root, dir, img_name))
      enhanced_img = enhance_image(img)
      imwrite(os.path.join(new_root, dir, img_name), enhanced_img)
```

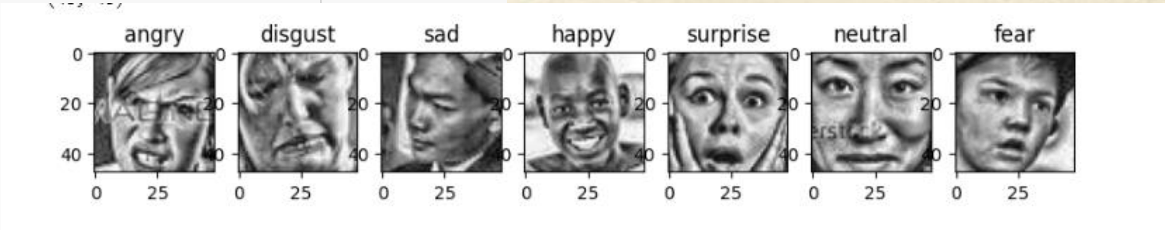


Preprocessing

- apply **CLAHE** to enhance image contrast



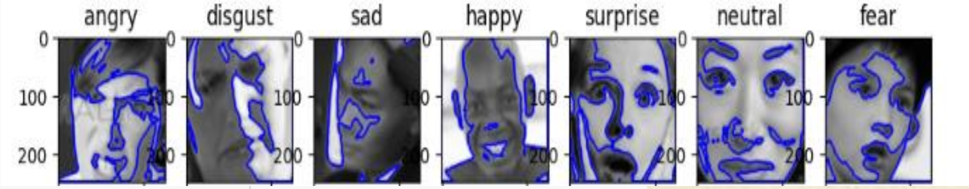
```
#function to enhance image using opencv
def enhance_image(img):
  #convert image to grayscale
  gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

  #apply CLAHE to enhance image contrast  (Contrast Limited Adaptive Histogram Equalization)
  clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
  enhanced_img = clahe.apply(gray_img)

  #convert image back to BGR
  enhanced_img = cv2.cvtColor(enhanced_img, cv2.COLOR_GRAY2BGR)

  #return enhanced image
  return enhanced_img

#create new dataset to store enhanced images
enhanced_root = '/content/drive/MyDrive/enhanced2'
os.makedirs(enhanced_root, exist_ok=True)

#iterate over each directory in original dataset
for dir in os.listdir(root):
  #create directory to store enhanced images for each class
  enhanced_dir = os.path.join(enhanced_root, dir)
  os.makedirs(enhanced_dir, exist_ok=True)

  #iterate over each image in directory
  for img_name in os.listdir(os.path.join(root, dir)):
    #load image
    img = imread(os.path.join(root, dir, img_name))

    #enhance image
    enhanced_img = enhance_image(img)
```



Segmentation

- We applied segmentation by using **findContours** tech against a thershold

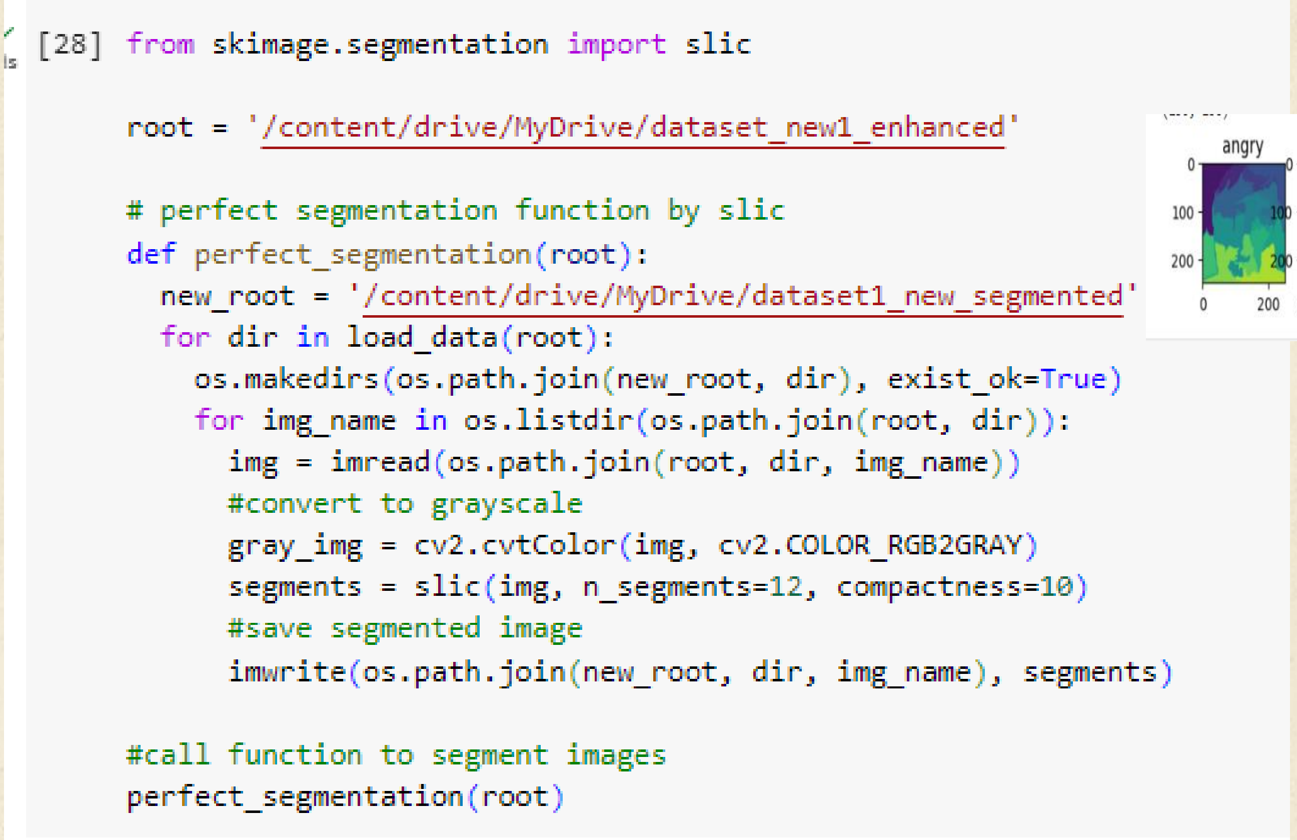
```
def perfect_segmentation(root):
  new_root = '/content/drive/MyDrive/dataset_new_segmented'
  for dir in load_data(root):
    os.makedirs(os.path.join(new_root, dir), exist_ok=True)
    for img_name in os.listdir(os.path.join(root, dir)):
      img = imread(os.path.join(root, dir, img_name))
      #convert to grayscale
      gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
      #apply thresholding
      ret, thresh = cv2.threshold(gray_img, 127, 255, 0)
      #find contours
      contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
      #draw contours on image
      cv2.drawContours(img, contours, -1, (255, 0, 0), 3)
      imwrite(os.path.join(new_root, dir, img_name), img)

#call function to segment images
perfect_segmentation(new_root)
```



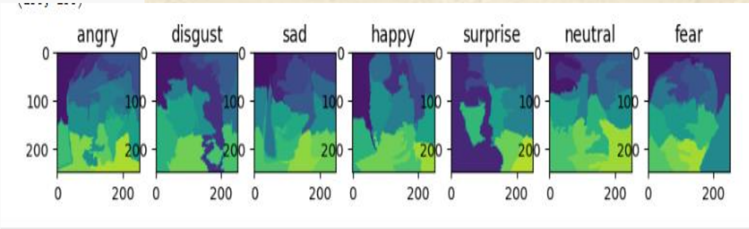
Segmentation

- segmentation function by **slic**

By controlling the number of segments and compactness, segmentation is applied to the images



```
[28] from skimage.segmentation import slic

root = '/content/drive/MyDrive/dataset_new1_enhanced'

# perfect segmentation function by slic
def perfect_segmentation(root):
  new_root = '/content/drive/MyDrive/dataset1_new_segmented'
  for dir in load_data(root):
    os.makedirs(os.path.join(new_root, dir), exist_ok=True)
    for img_name in os.listdir(os.path.join(root, dir)):
      img = imread(os.path.join(root, dir, img_name))
      #convert to grayscale
      gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
      segments = slic(img, n_segments=12, compactness=10)
      #save segmented image
      imwrite(os.path.join(new_root, dir, img_name), segments)

#call function to segment images
perfect_segmentation(root)
```



Feature extraction

- We applied feature extraction through various algorithms. First, we used the **GLCM** technique to calculate contrast , 'homogeneity' , 'energy' and correlation' for each dataset image.

- Also , using **SIFT** as key point detector and descriptor

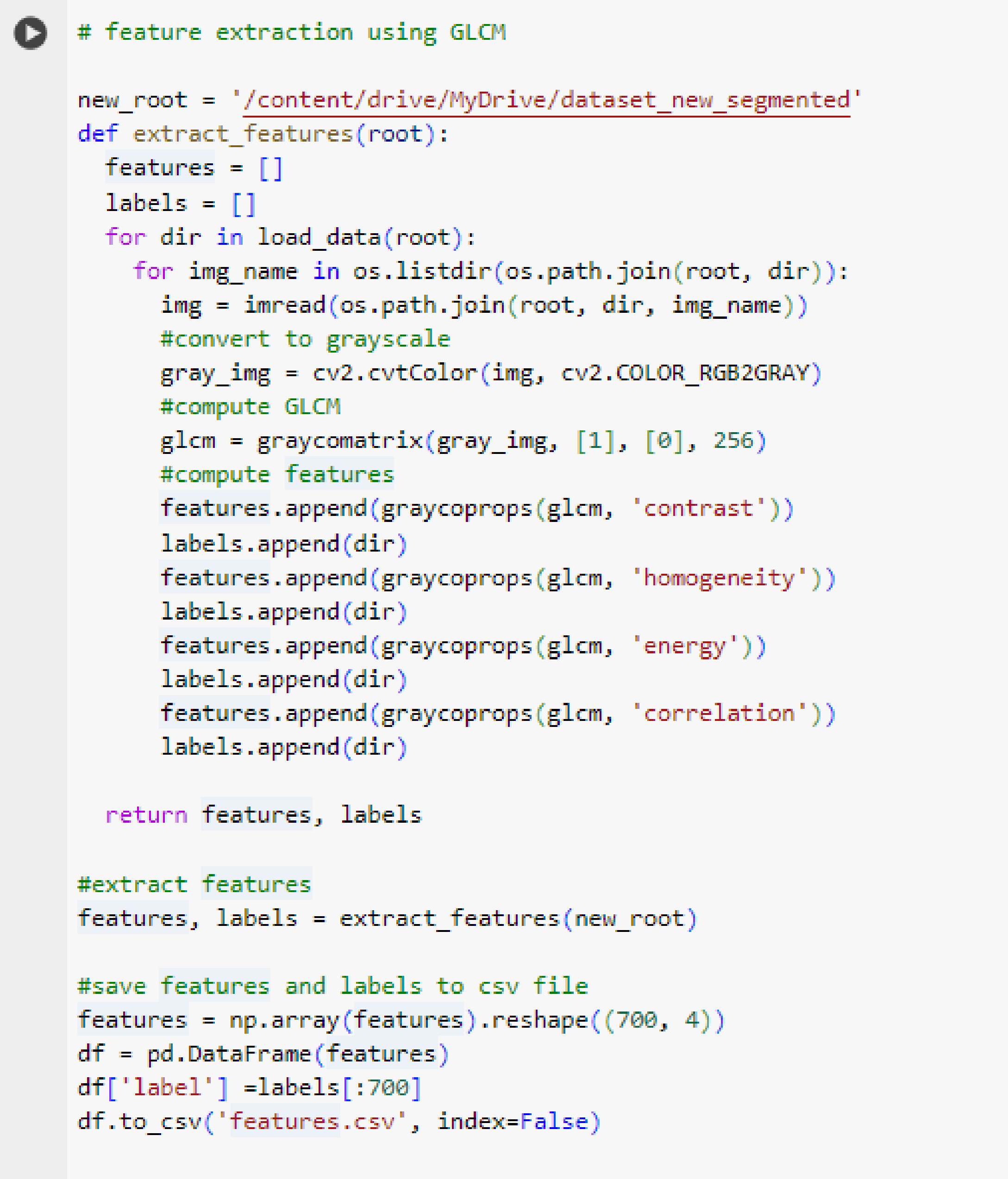
It captures details about gradient magnitude and orientation in the neighborhood of the key point.

- By calculation **regional features**

(area , perimeter , compactness)

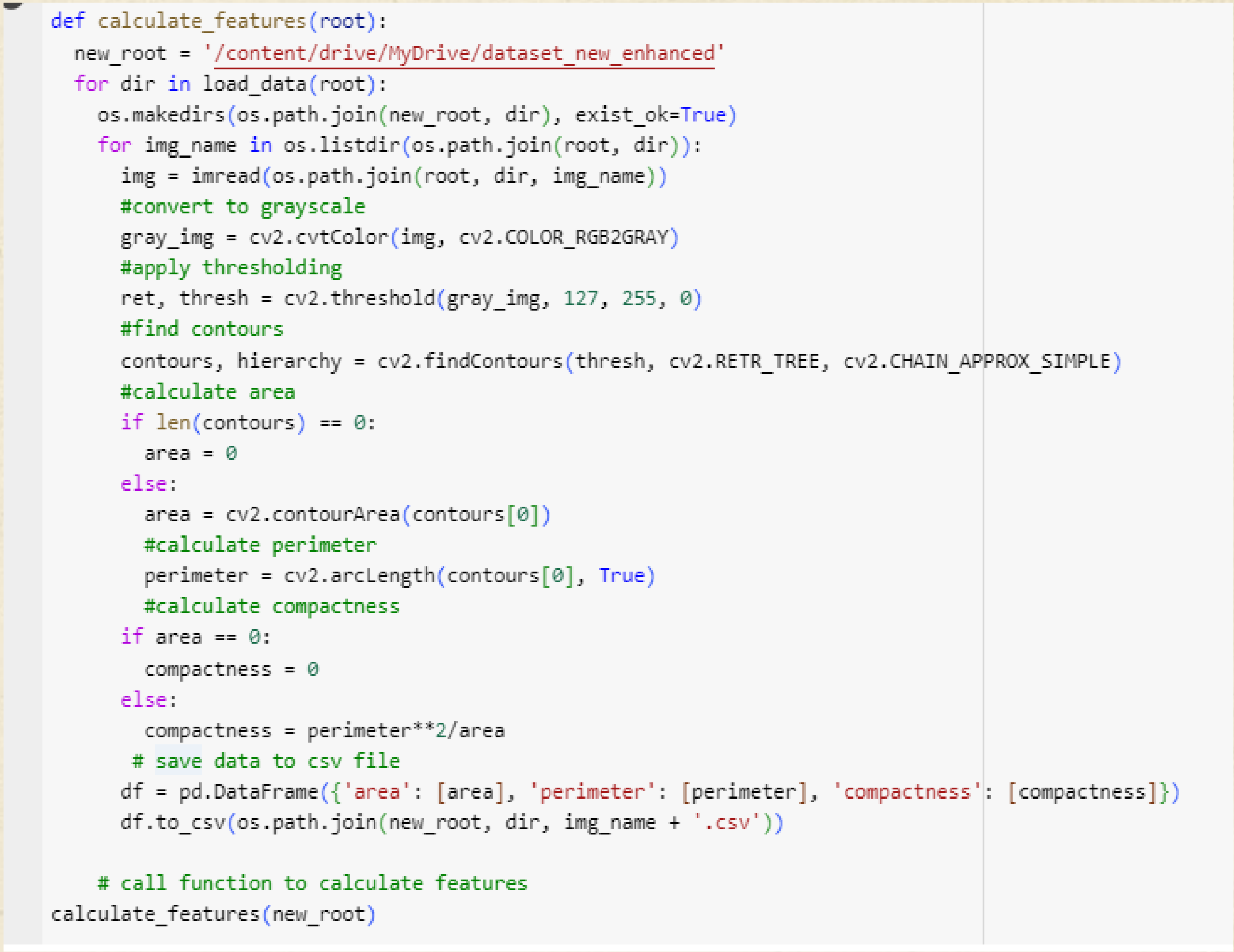
✓ All feature description stored in csv files

GLCM



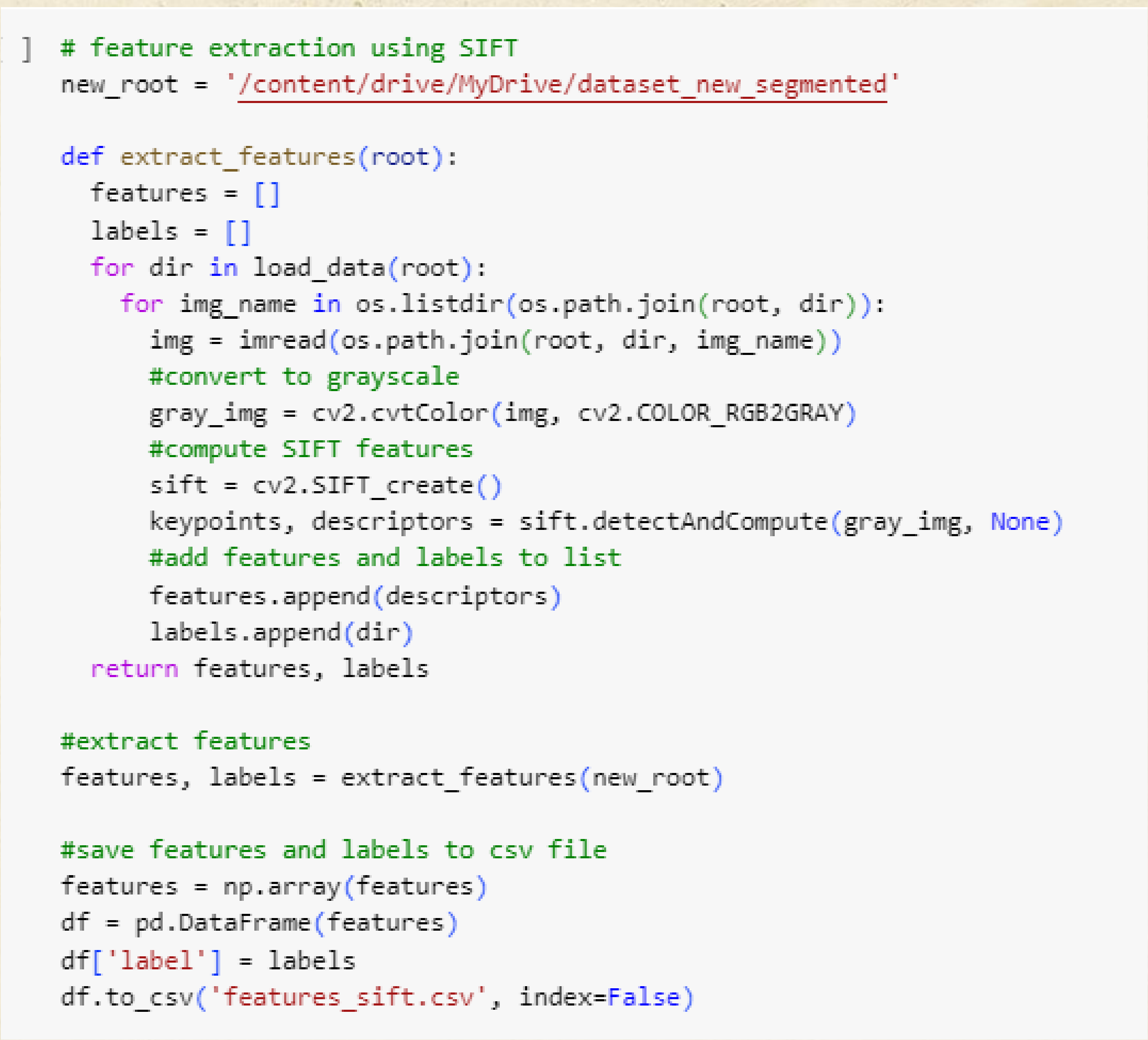
```python
# feature extraction using GLCM

new_root = '/content/drive/MyDrive/dataset_new_segmented'
def extract_features(root):
  features = []
  labels = []
  for dir in load_data(root):
    for img_name in os.listdir(os.path.join(root, dir)):
      img = imread(os.path.join(root, dir, img_name))
      #convert to grayscale
      gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
      #compute GLCM
      glcm = graycomatrix(gray_img, [1], [0], 256)
      #compute features
      features.append(graycoprops(glcm, 'contrast'))
      labels.append(dir)
      features.append(graycoprops(glcm, 'homogeneity'))
      labels.append(dir)
      features.append(graycoprops(glcm, 'energy'))
      labels.append(dir)
      features.append(graycoprops(glcm, 'correlation'))
      labels.append(dir)

  return features, labels

#extract features
features, labels = extract_features(new_root)

#save features and labels to csv file
features = np.array(features).reshape((700, 4))
df = pd.DataFrame(features)
df['label'] =labels[:700]
df.to_csv('features.csv', index=False)
```

Regional features

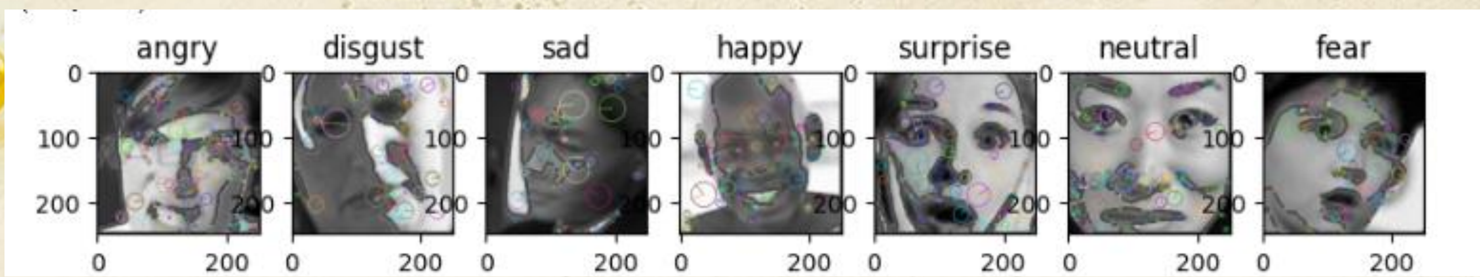


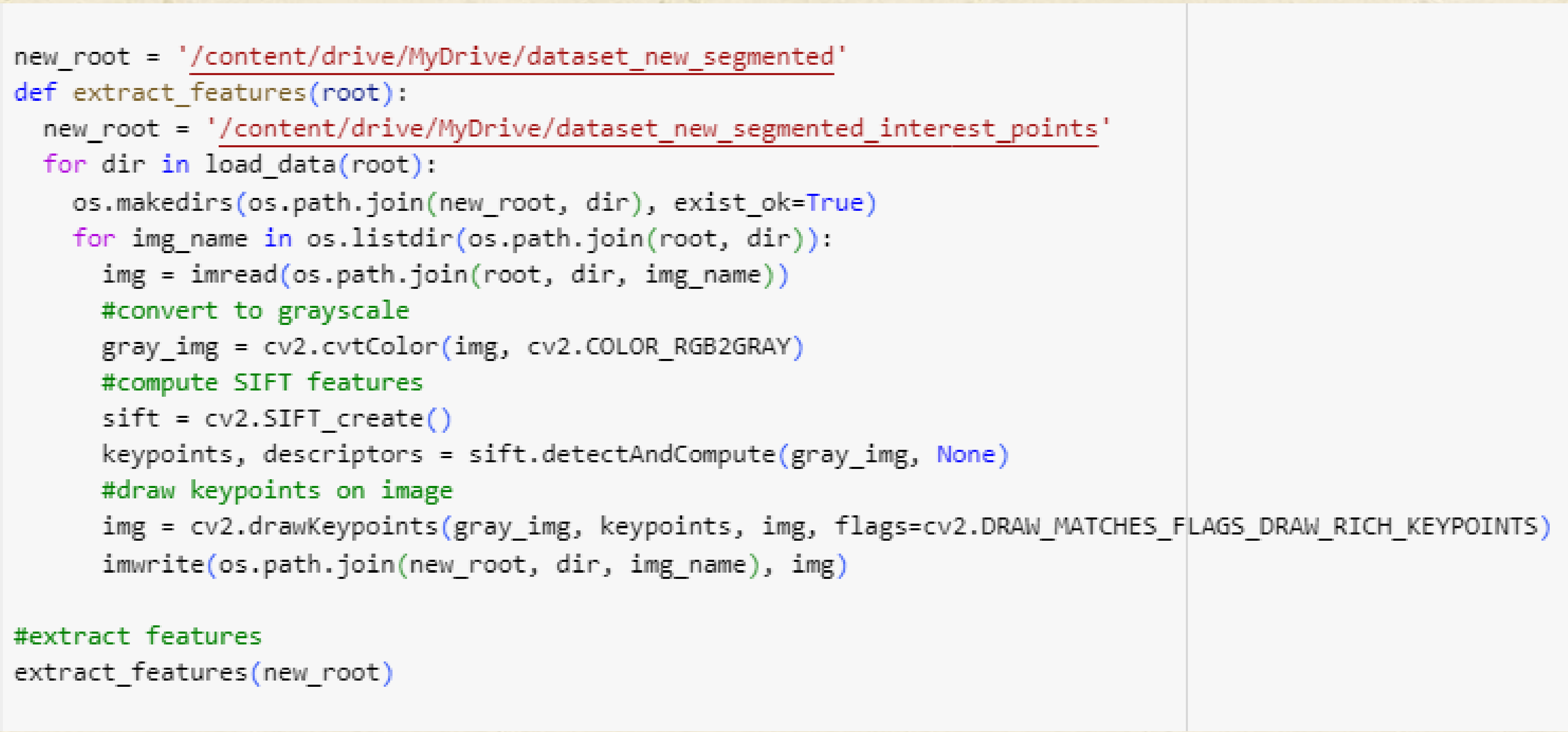
```
def calculate_features(root):
  new_root = '/content/drive/MyDrive/dataset_new_enhanced'
  for dir in load_data(root):
    os.makedirs(os.path.join(new_root, dir), exist_ok=True)
    for img_name in os.listdir(os.path.join(root, dir)):
      img = imread(os.path.join(root, dir, img_name))
      #convert to grayscale
      gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
      #apply thresholding
      ret, thresh = cv2.threshold(gray_img, 127, 255, 0)
      #find contours
      contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
      #calculate area
      if len(contours) == 0:
        area = 0
      else:
        area = cv2.contourArea(contours[0])
        #calculate perimeter
        perimeter = cv2.arcLength(contours[0], True)
        #calculate compactness
      if area == 0:
        compactness = 0
      else:
        compactness = perimeter**2/area
       # save data to csv file
      df = pd.DataFrame({'area': [area], 'perimeter': [perimeter], 'compactness': [compactness]})
      df.to_csv(os.path.join(new_root, dir, img_name + '.csv'))

    # call function to calculate features
calculate_features(new_root)
```

SIFT



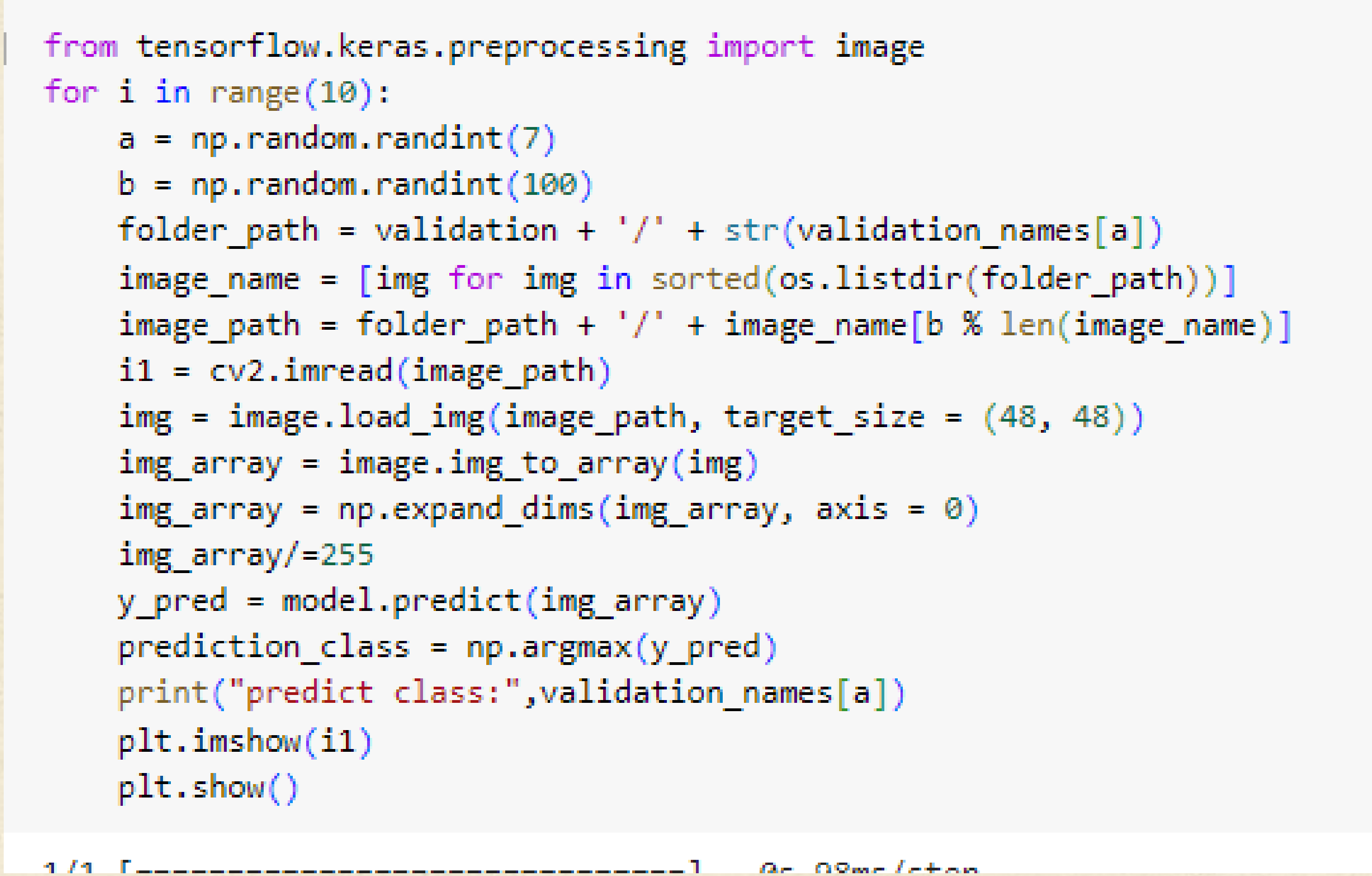
```
# feature extraction using SIFT
new_root = '/content/drive/MyDrive/dataset_new_segmented'

def extract_features(root):
  features = []
  labels = []
  for dir in load_data(root):
    for img_name in os.listdir(os.path.join(root, dir)):
      img = imread(os.path.join(root, dir, img_name))
      #convert to grayscale
      gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
      #compute SIFT features
      sift = cv2.SIFT_create()
      keypoints, descriptors = sift.detectAndCompute(gray_img, None)
      #add features and labels to list
      features.append(descriptors)
      labels.append(dir)
  return features, labels

#extract features
features, labels = extract_features(new_root)

#save features and labels to csv file
features = np.array(features)
df = pd.DataFrame(features)
df['label'] = labels
df.to_csv('features_sift.csv', index=False)
```





```
new_root = '/content/drive/MyDrive/dataset_new_segmented'
def extract_features(root):
  new_root = '/content/drive/MyDrive/dataset_new_segmented_interest_points'
  for dir in load_data(root):
    os.makedirs(os.path.join(new_root, dir), exist_ok=True)
    for img_name in os.listdir(os.path.join(root, dir)):
      img = imread(os.path.join(root, dir, img_name))
      #convert to grayscale
      gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
      #compute SIFT features
      sift = cv2.SIFT_create()
      keypoints, descriptors = sift.detectAndCompute(gray_img, None)
      #draw keypoints on image
      img = cv2.drawKeypoints(gray_img, keypoints, img, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
      imwrite(os.path.join(new_root, dir, img_name), img)

#extract features
extract_features(new_root)
```

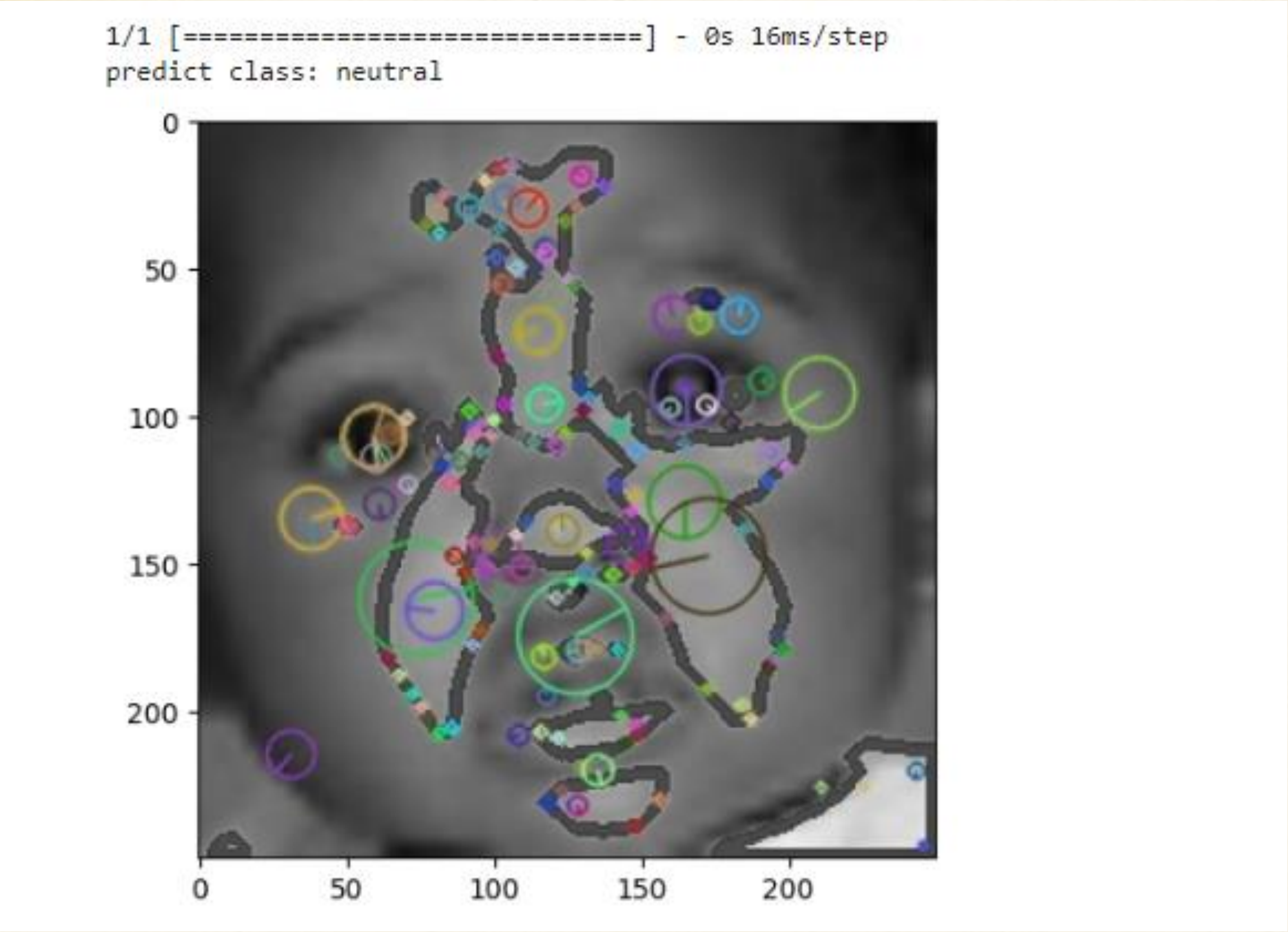
Classification



```
from tensorflow.keras.preprocessing import image
for i in range(10):
    a = np.random.randint(7)
    b = np.random.randint(100)
    folder_path = validation + '/' + str(validation_names[a])
    image_name = [img for img in sorted(os.listdir(folder_path))]
    image_path = folder_path + '/' + image_name[b % len(image_name)]
    i1 = cv2.imread(image_path)
    img = image.load_img(image_path, target_size = (48, 48))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis = 0)
    img_array/=255
    y_pred = model.predict(img_array)
    prediction_class = np.argmax(y_pred)
    print("predict class:",validation_names[a])
    plt.imshow(i1)
    plt.show()
```

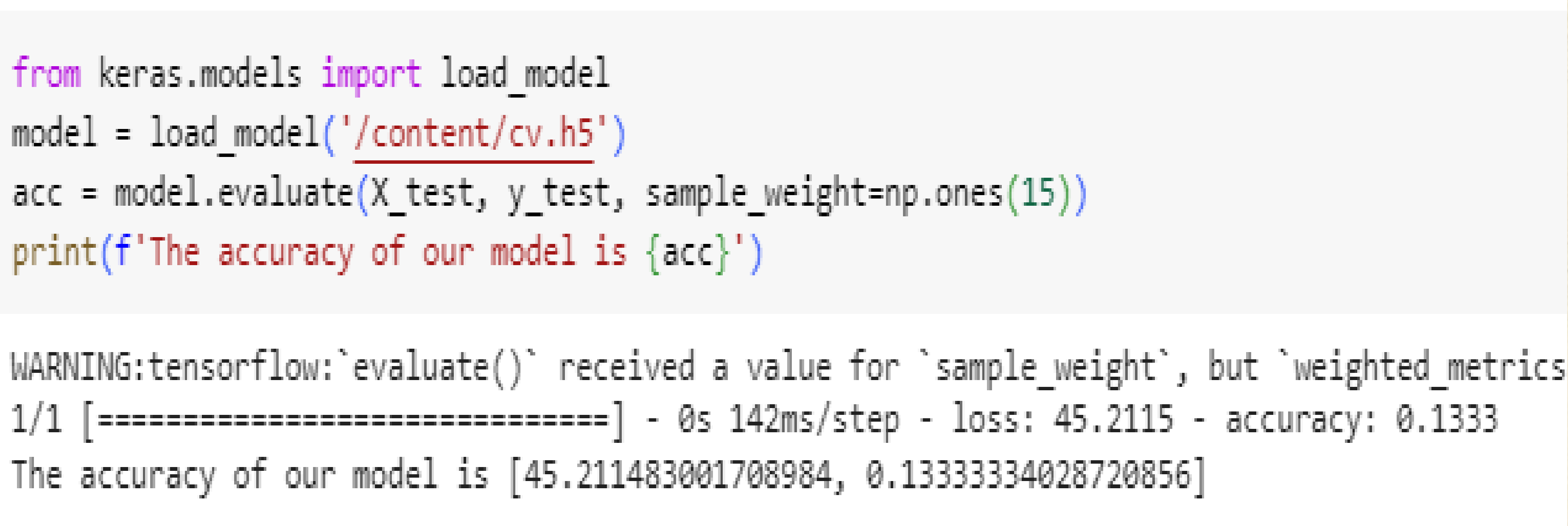
```
1/1 [==============================] - 0s 98ms/step
```

Classification



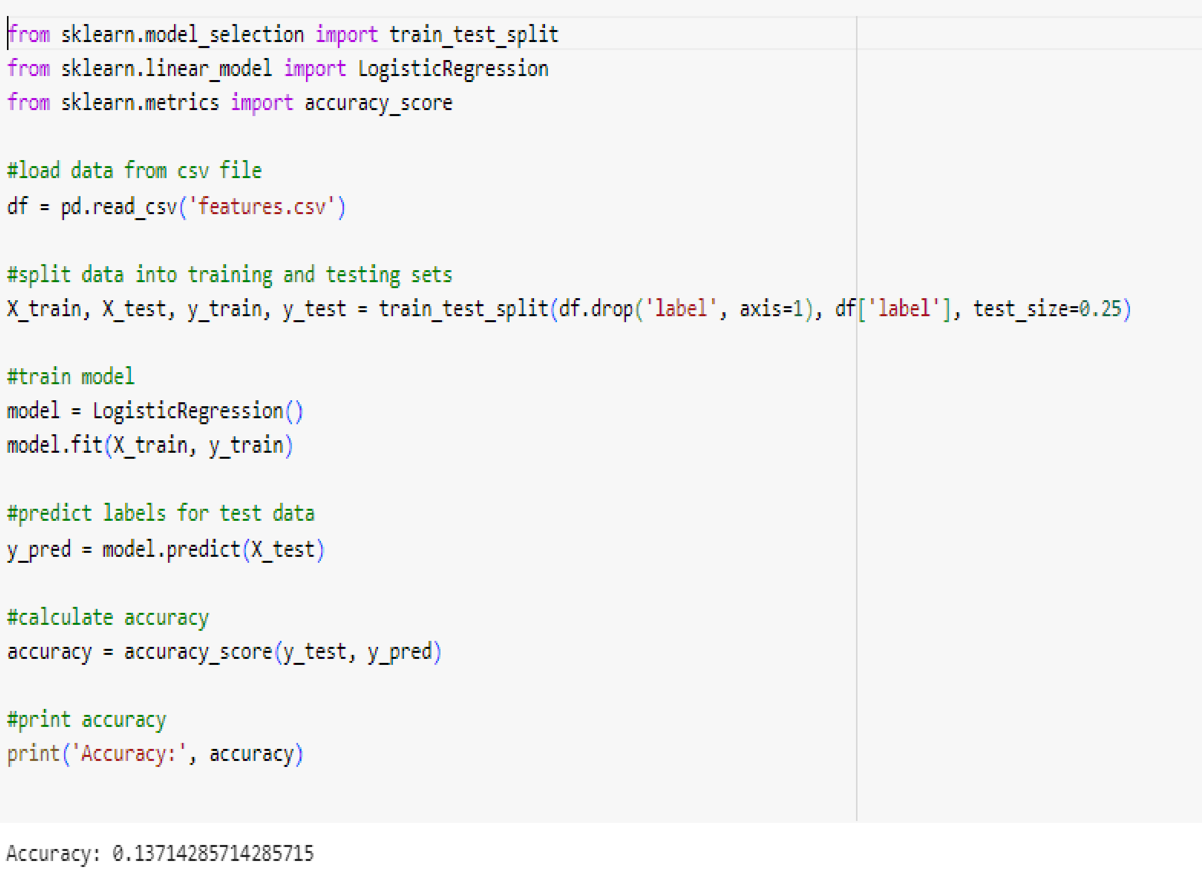
```
1/1 [==============================] - 0s 16ms/step
predict class: neutral
```

Classification



```
from keras.models import load_model
model = load_model('/content/cv.h5')
acc = model.evaluate(X_test, y_test, sample_weight=np.ones(15))
print(f'The accuracy of our model is {acc}')
```

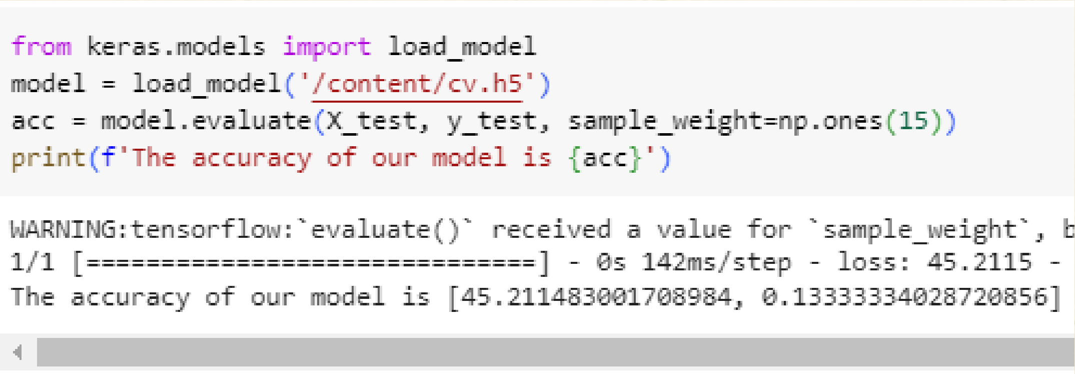
```
WARNING:tensorflow:`evaluate()` received a value for `sample_weight`, but `weighted_metrics
1/1 [==============================] - 0s 142ms/step - loss: 45.2115 - accuracy: 0.1333
The accuracy of our model is [45.211483001708984, 0.13333334028720856]
```



```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

#load data from csv file
df = pd.read_csv('features.csv')

#split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df.drop('label', axis=1), df['label'], test_size=0.25)

#train model
model = LogisticRegression()
model.fit(X_train, y_train)

#predict labels for test data
y_pred = model.predict(X_test)

#calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

#print accuracy
print('Accuracy:', accuracy)
```

```
Accuracy: 0.13714285714285715
```

GLCM accuracy score



```
from keras.models import load_model
model = load_model('/content/cv.h5')
acc = model.evaluate(X_test, y_test, sample_weight=np.ones(15))
print(f'The accuracy of our model is {acc}')
```

```
WARNING:tensorflow:`evaluate()` received a value for `sample_weight`, b
1/1 [==============================] - 0s 142ms/step - loss: 45.2115 -
The accuracy of our model is [45.211483001708984, 0.13333334028720856]
```

model accuracy score

THANK
YOU!