Project_2_Final_Working_Version_test

March 3, 2025

Utilizing Various Machine Learning Models to Classify Emotions Based Off Facial Expressions - Cassandra Morgan

For my project I decided to see if I could create a simple machine learning model to correctly classify the emotions displayed on human faces through analyzing and categorizing images of people and splitting them into 6 distinct categories: angry, fearful, happy, neutral, sad or surprised.

For this I used a dataset from Kaggle: https://www.kaggle.com/datasets/ananthu017/emotion-detection-fer?resource=download

I dropped the emotion category called "disgust" from the dataset and removed it from my list of classifiers in order to simplify the models. I also removed that category because it had only 1k observations, and the other datasets all had 3k+. I wanted to ensure that each category had a similar number of observations, but I did not want to limit myself to 1k observations per category, so I removed images classified as "disgust" from the dataset. For my models I used a train-test split for validation.

Analyzing the facial expressions of people in images to categorize their emotional state using machine learning has useful applications for analyzing images to make educational material for children or autistic people who struggle to identify emotions based off facial expressions. I used three separate models for this: a neural net, a random forest, and a kmeans clustering technique.

First I installed the necessary packages for creating my models and reading in the image data from the directories that the data were structured into.

For the first section, I chose to create a neural net.

Neural Net:

Next I imported the necessary modules.

[192]:
```
import tensorflow as tf
from tensorflow.keras import layers
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import glob
from sklearn.preprocessing import LabelEncoder
import pandas as pd
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

Here, I created arrays to hold the names and associated index numbers.

[194]:
```
class_names = ['Angry', 'Fearful', 'Happy', 'Neutral', 'Sad', 'Surprised']
class_names_numbers = [int(0),int(1),int(2),int(3),int(4),int(5)]
```

Next I read in the image data from the directories. To do this, I used glob and cv2 to read in images file by file. The images were already randomly sorted within each directory, so I didn't have to sort them again. I truncated each array of images to be 3000 images long. I did this so that none of the categories would have more data than the rest. In the original data the happy category had 4k more images than the smallest category. It is known that neural nets do not do as well on unbalanced data, and I wanted to ensure it was not biased towards the happy category.

[196]:
```
angry_image_list = []
for filename in glob.glob('D:/Adult Stuff/DAT_402/Project_2/train/angry/*.png'):
    angry_im = cv2.imread(filename)
    angry_image_list.append(angry_im)
angry_image__array = np.array(angry_image_list)
angry_image__array = angry_image__array[:3000]
len(angry_image__array)
```

[196]: 3000

[197]:
```
fearful_image_list = []
for filename in glob.glob('D:/Adult Stuff/DAT_402/Project_2/train/fearful/*.
 ↪png'):
    fearful_im= cv2.imread(filename)
    fearful_image_list.append(fearful_im)
fearful_image__array = np.array(fearful_image_list)
fearful_image__array = fearful_image__array[:3000]
len(fearful_image__array)
```

[197]: 3000

[198]:
```
happy_image_list = []
for filename in glob.glob('D:/Adult Stuff/DAT_402/Project_2/train/happy/*.png'):
    happy_im= cv2.imread(filename)
    happy_image_list.append(happy_im)
happy_image__array = np.array(happy_image_list)
happy_image__array = happy_image__array[:3000]
len(happy_image__array)
```

[198]: 3000

[199]:
```
neutral_image_list = []
for filename in glob.glob('D:/Adult Stuff/DAT_402/Project_2/train/neutral/*.
 ↪png'):
    neutral_im= cv2.imread(filename)
    neutral_image_list.append(neutral_im)
```

```
neutral_image__array = np.array(neutral_image_list)
neutral_image__array = neutral_image__array[:3000]
len(neutral_image__array)
```

[199]: 3000

[200]:
```
sad_image_list = []
for filename in glob.glob('D:/Adult Stuff/DAT_402/Project_2/train/sad/*.png'):
    sad_im= cv2.imread(filename)
    sad_image_list.append(sad_im)
sad_image__array = np.array(sad_image_list)
sad_image__array = sad_image__array[:3000]
len(sad_image__array)
```

[200]: 3000

[201]:
```
surprised_image_list = []
for filename in glob.glob('D:/Adult Stuff/DAT_402/Project_2/train/surprised/*.
 ↪png'):
    surprised_im= cv2.imread(filename)
    surprised_image_list.append(surprised_im)
surprised_image__array = np.array(surprised_image_list)
surprised_image__array = surprised_image__array[:3000]
len(surprised_image__array)
```

[201]: 3000

Combining all the training data into one training set for use with models.

[203]:
```
train_set = np.concatenate((angry_image__array, fearful_image__array,␣
 ↪happy_image__array, neutral_image__array, sad_image__array,␣
 ↪surprised_image__array))
train_set_labels = np.empty(18000)
print(len(train_set))
print(len(train_set_labels))
```

```
18000
18000
```

Assigning the true labels of emotions for the training data.

[205]:
```
index_break_0 = len(angry_image__array)
index_break_1 =  index_break_0+len(fearful_image__array)
index_break_2 = index_break_1 + len(happy_image__array)
index_break_3 = index_break_2 + len(neutral_image__array)
index_break_4 = index_break_3 + len(sad_image__array)
index_break_5 = index_break_4 + len(surprised_image__array)
```

```
train_set_labels[0:index_break_0] = 0
train_set_labels[index_break_0:index_break_1] = 1
train_set_labels[index_break_1:index_break_2] = 2
train_set_labels[index_break_2:index_break_3] = 3
train_set_labels[index_break_3:index_break_4] = 4
train_set_labels[index_break_4:index_break_5] = 5
```

I looked at the shape of the training set.

[207]:
```
print('number of images:',len(train_set))

#checking type of image
print('type:', type(train_set[0]))

#shape of the first image
print('shape:', train_set[0].shape)

#what are minimum and maximum values in the entire train_images
print('image value range:', (np.min(train_set), np.max(train_set)))
```

```
number of images: 18000
type: <class 'numpy.ndarray'>
shape: (48, 48, 3)
image value range: (0, 255)
```

[208]:
```
print('number of labels:', len(train_set_labels))
print('type:',type(train_set))
print('shape:', train_set_labels.shape)
print('set of label values:', set(train_set_labels))
```

```
number of labels: 18000
type: <class 'numpy.ndarray'>
shape: (18000,)
set of label values: {0.0, 1.0, 2.0, 3.0, 4.0, 5.0}
```

Next, I read in the test data and created arrays from the images.

[210]:
```
angry_image_list_test = []
for filename in glob.glob('D:/Adult Stuff/DAT_402/Project_2/test/angry/*.png'):
    angry_im_test = cv2.imread(filename)
    angry_image_list_test.append(angry_im_test)
angry_image__array_test = np.array(angry_image_list_test)
len(angry_image__array_test)
```

[210]:
```
958
```

[211]:
```
fearful_image_list_test = []
for filename in glob.glob('D:/Adult Stuff/DAT_402/Project_2/test/fearful/*.
 ↪png'):
```

```
    fearful_im_test = cv2.imread(filename)
    fearful_image_list_test.append(fearful_im_test)
fearful_image__array_test = np.array(fearful_image_list_test)
len(fearful_image__array_test)
```

[211]: 1024

[212]:
```
happy_image_list_test = []
for filename in glob.glob('D:/Adult Stuff/DAT_402/Project_2/test/happy/*.png'):
    happy_im_test = cv2.imread(filename)
    happy_image_list_test.append(happy_im_test)
happy_image__array_test = np.array(happy_image_list_test)
len(happy_image__array_test)
```

[212]: 1774

[213]:
```
neutral_image_list_test = []
for filename in glob.glob('D:/Adult Stuff/DAT_402/Project_2/test/neutral/*.
 ↪png'):
    neutral_im_test = cv2.imread(filename)
    neutral_image_list_test.append(neutral_im_test)
neutral_image__array_test = np.array(neutral_image_list_test)
len(neutral_image__array_test)
```

[213]: 1233

[214]:
```
sad_image_list_test = []
for filename in glob.glob('D:/Adult Stuff/DAT_402/Project_2/test/sad/*.png'):
    sad_im_test = cv2.imread(filename)
    sad_image_list_test.append(sad_im_test)
sad_image__array_test = np.array(sad_image_list_test)
len(sad_image__array_test)
```

[214]: 1247

[215]:
```
surprised_image_list_test = []
for filename in glob.glob('D:/Adult Stuff/DAT_402/Project_2/test/surprised/*.
 ↪png'):
    surprised_im_test = cv2.imread(filename)
    surprised_image_list_test.append(surprised_im_test)
surprised_image__array_test = np.array(surprised_image_list_test)
len(surprised_image__array_test)
```

[215]: 831

Combining the test data.

[217]:
```
test_set = np.concatenate((angry_image__array_test, fearful_image__array_test,␣
 ↪happy_image__array_test, neutral_image__array_test, sad_image__array_test,␣
 ↪surprised_image__array_test))
test_set_labels = np.zeros(7067, dtype=int)
print(len(test_set))
print(len(test_set_labels))
```

```
7067
7067
```

Assigning the true labels of emotions for the training data.

[219]:
```
index_break_0_test = len(angry_image__array_test)
index_break_1_test =  index_break_0_test+len(fearful_image__array_test)
index_break_2_test = index_break_1_test + len(happy_image__array_test)
index_break_3_test = index_break_2_test + len(neutral_image__array_test)
index_break_4_test = index_break_3_test + len(sad_image__array_test)
index_break_5_test = index_break_4_test + len(surprised_image__array_test)


test_set_labels[0:index_break_0_test] = 0
test_set_labels[index_break_0_test:index_break_1_test] = 1
test_set_labels[index_break_1_test:index_break_2_test] = 2
test_set_labels[index_break_2_test:index_break_3_test] = 3
test_set_labels[index_break_3_test:index_break_4_test] = 4
test_set_labels[index_break_4_test:index_break_5_test] = 5
```

[220]:
```
print('test_images shape:', test_set.shape)
print('image value range:', (np.min(test_set), np.max(test_set)))

print('test_labels shape:', test_set_labels.shape)
print('set of label values', set(test_set_labels))
```

```
test_images shape: (7067, 48, 48, 3)
image value range: (0, 255)
test_labels shape: (7067,)
set of label values {0, 1, 2, 3, 4, 5}
```

[221]:
```
print('test_images shape:', train_set.shape)
print('image value range:', (np.min(train_set), np.max(train_set)))

print('test_labels shape:', train_set_labels.shape)
print('set of label values', set(train_set_labels))
```

```
test_images shape: (18000, 48, 48, 3)
image value range: (0, 255)
test_labels shape: (18000,)
set of label values {0.0, 1.0, 2.0, 3.0, 4.0, 5.0}
```

Concepts from: HW4

This next code chunk is for plotting an image and its estimate for the label along with the true label. This code works for neural nets.

[223]:
```
def plot_image(i, predictions_array, true_label, img):
    true_label, img = true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img, cmap=plt.cm.binary)

    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
        color = (0,0.6,0)
    else:
        color = 'red'

    plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)

def plot_value_array(i, predictions_array, true_label):
    true_label = true_label[i]
    plt.grid(False)
    plt.xticks(range(6))
    plt.yticks([])
    thisplot = plt.bar(range(6), predictions_array, color=(0.7,0.7,0.7)) ␣
 ↪#paint all bars in gray
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)

    thisplot[predicted_label].set_color('red') #over-paint the predicted (i.e.␣
 ↪tallest) bar in red
    thisplot[true_label].set_color((0,0.8,0)) #over-paint the bar for the true␣
 ↪label in green
```

This code does the same plotting procedure but for non-neural nets.

[225]:
```
def plot_image_2 (i, predictions_array, true_label_array, img):
    true_label, img = true_label_array[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img, cmap=plt.cm.binary)

    if predictions_array[i] == true_label_array[i]:
```

```
        color = (0,0.6,0)
    else:
        color = 'red'

    plt.xlabel("{} ({})".format(class_names[int(predictions_array[i])],
                                class_names[true_label_array[i]]),
                                color=color)

def plot_value_array_2(i, predictions_array, true_label):
    true_label = true_label[i]
    plt.grid(False)
    plt.xticks(range(6))
    plt.yticks([])
    thisplot = plt.bar(range(6), predictions_array, color=(0.7,0.7,0.7))
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)

    thisplot[predicted_label].set_color('red') #over-paint the predicted (i.e.␣
 ↪tallest) bar in red
    thisplot[true_label].set_color((0,0.8,0)) #over-paint the bar for the true␣
 ↪label in green
```

I scaled the training and test data because it has to be between 0 and 1 for the neural net. The values ranged from 0 to 255 before because they were describing pixels.

[227]:
```
train_set_nn = train_set / 255
test_set_nn = test_set / 255
```

Next, I built the model for the neural net. I used a convolution neural net with relu for the first two activation functions, and my ouput layer's activation function was softmax.

[229]:
```
np.random.seed(234) #set the seed in numpy
tf.random.set_seed(2345)  #set the seed in tensorflow



mynet = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48,␣
 ↪3)),
  tf.keras.layers.MaxPooling2D((2, 2)),
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(6, activation='softmax')
], name = 'mynet')
```

```
C:\Users\Damian\anaconda3\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
```

```
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

For my optimizer during the compiling step, I used 'adam' because it is good for image classification.

[231]:
```
mynet.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Next I fit the model for 1 through 50 epochs and recorded the accuracy, loss, validation accuracy, and validation loss.

[233]:
```
history = mynet.fit(x=train_set_nn, y=train_set_labels,␣
 ↪validation_data=(test_set_nn, test_set_labels), epochs=50, batch_size=32)
```

```
Epoch 1/50
563/563             5s 8ms/step -
accuracy: 0.2712 - loss: 1.7089 - val_accuracy: 0.4170 - val_loss: 1.5009
Epoch 2/50
563/563             5s 8ms/step -
accuracy: 0.4233 - loss: 1.4629 - val_accuracy: 0.4562 - val_loss: 1.3912
Epoch 3/50
563/563             4s 7ms/step -
accuracy: 0.4712 - loss: 1.3524 - val_accuracy: 0.4699 - val_loss: 1.3591
Epoch 4/50
563/563             5s 9ms/step -
accuracy: 0.5117 - loss: 1.2707 - val_accuracy: 0.4821 - val_loss: 1.3559
Epoch 5/50
563/563             5s 9ms/step -
accuracy: 0.5424 - loss: 1.2027 - val_accuracy: 0.4838 - val_loss: 1.3646
Epoch 6/50
563/563             5s 8ms/step -
accuracy: 0.5712 - loss: 1.1388 - val_accuracy: 0.4824 - val_loss: 1.3869
Epoch 7/50
563/563             4s 7ms/step -
accuracy: 0.5988 - loss: 1.0822 - val_accuracy: 0.4814 - val_loss: 1.4347
Epoch 8/50
563/563             5s 8ms/step -
accuracy: 0.6214 - loss: 1.0233 - val_accuracy: 0.4790 - val_loss: 1.4907
Epoch 9/50
563/563             5s 9ms/step -
accuracy: 0.6434 - loss: 0.9718 - val_accuracy: 0.4788 - val_loss: 1.4886
Epoch 10/50
563/563             4s 8ms/step -
accuracy: 0.6643 - loss: 0.9257 - val_accuracy: 0.4781 - val_loss: 1.5418
Epoch 11/50
563/563             5s 9ms/step -
accuracy: 0.6804 - loss: 0.8815 - val_accuracy: 0.4704 - val_loss: 1.6125
```

```
Epoch 12/50
563/563             5s 9ms/step -
accuracy: 0.6922 - loss: 0.8485 - val_accuracy: 0.4590 - val_loss: 1.7193
Epoch 13/50
563/563             5s 8ms/step -
accuracy: 0.7015 - loss: 0.8215 - val_accuracy: 0.4568 - val_loss: 1.7668
Epoch 14/50
563/563             4s 8ms/step -
accuracy: 0.7178 - loss: 0.7900 - val_accuracy: 0.4559 - val_loss: 1.8342
Epoch 15/50
563/563             4s 7ms/step -
accuracy: 0.7312 - loss: 0.7573 - val_accuracy: 0.4609 - val_loss: 1.8674
Epoch 16/50
563/563             5s 8ms/step -
accuracy: 0.7412 - loss: 0.7307 - val_accuracy: 0.4619 - val_loss: 1.9476
Epoch 17/50
563/563             4s 7ms/step -
accuracy: 0.7443 - loss: 0.7108 - val_accuracy: 0.4631 - val_loss: 1.9936
Epoch 18/50
563/563             5s 8ms/step -
accuracy: 0.7536 - loss: 0.6880 - val_accuracy: 0.4648 - val_loss: 2.0277
Epoch 19/50
563/563             5s 9ms/step -
accuracy: 0.7627 - loss: 0.6682 - val_accuracy: 0.4646 - val_loss: 2.0549
Epoch 20/50
563/563             4s 8ms/step -
accuracy: 0.7736 - loss: 0.6441 - val_accuracy: 0.4643 - val_loss: 2.1266
Epoch 21/50
563/563             5s 8ms/step -
accuracy: 0.7850 - loss: 0.6209 - val_accuracy: 0.4641 - val_loss: 2.2027
Epoch 22/50
563/563             4s 8ms/step -
accuracy: 0.7898 - loss: 0.6041 - val_accuracy: 0.4573 - val_loss: 2.2842
Epoch 23/50
563/563             4s 8ms/step -
accuracy: 0.7972 - loss: 0.5795 - val_accuracy: 0.4602 - val_loss: 2.3478
Epoch 24/50
563/563             5s 9ms/step -
accuracy: 0.8054 - loss: 0.5580 - val_accuracy: 0.4604 - val_loss: 2.4312
Epoch 25/50
563/563             5s 8ms/step -
accuracy: 0.8108 - loss: 0.5455 - val_accuracy: 0.4589 - val_loss: 2.5032
Epoch 26/50
563/563             5s 8ms/step -
accuracy: 0.8184 - loss: 0.5275 - val_accuracy: 0.4555 - val_loss: 2.5762
Epoch 27/50
563/563             4s 8ms/step -
accuracy: 0.8200 - loss: 0.5145 - val_accuracy: 0.4514 - val_loss: 2.6467
```

```
Epoch 28/50
563/563             5s 9ms/step -
accuracy: 0.8252 - loss: 0.5006 - val_accuracy: 0.4498 - val_loss: 2.7316
Epoch 29/50
563/563             5s 8ms/step -
accuracy: 0.8295 - loss: 0.4871 - val_accuracy: 0.4494 - val_loss: 2.7978
Epoch 30/50
563/563             5s 8ms/step -
accuracy: 0.8314 - loss: 0.4809 - val_accuracy: 0.4438 - val_loss: 2.9024
Epoch 31/50
563/563             5s 8ms/step -
accuracy: 0.8311 - loss: 0.4745 - val_accuracy: 0.4450 - val_loss: 2.9567
Epoch 32/50
563/563             5s 8ms/step -
accuracy: 0.8395 - loss: 0.4610 - val_accuracy: 0.4470 - val_loss: 3.0703
Epoch 33/50
563/563             5s 8ms/step -
accuracy: 0.8471 - loss: 0.4440 - val_accuracy: 0.4436 - val_loss: 3.1632
Epoch 34/50
563/563             5s 8ms/step -
accuracy: 0.8473 - loss: 0.4348 - val_accuracy: 0.4354 - val_loss: 3.3116
Epoch 35/50
563/563             5s 8ms/step -
accuracy: 0.8572 - loss: 0.4161 - val_accuracy: 0.4354 - val_loss: 3.3977
Epoch 36/50
563/563             4s 8ms/step -
accuracy: 0.8600 - loss: 0.4040 - val_accuracy: 0.4329 - val_loss: 3.4757
Epoch 37/50
563/563             4s 8ms/step -
accuracy: 0.8627 - loss: 0.3912 - val_accuracy: 0.4365 - val_loss: 3.5489
Epoch 38/50
563/563             4s 7ms/step -
accuracy: 0.8655 - loss: 0.3868 - val_accuracy: 0.4413 - val_loss: 3.5897
Epoch 39/50
563/563             4s 8ms/step -
accuracy: 0.8647 - loss: 0.3826 - val_accuracy: 0.4411 - val_loss: 3.7052
Epoch 40/50
563/563             6s 10ms/step -
accuracy: 0.8610 - loss: 0.3829 - val_accuracy: 0.4337 - val_loss: 3.8035
Epoch 41/50
563/563             4s 7ms/step -
accuracy: 0.8662 - loss: 0.3703 - val_accuracy: 0.4363 - val_loss: 3.9296
Epoch 42/50
563/563             6s 10ms/step -
accuracy: 0.8690 - loss: 0.3648 - val_accuracy: 0.4336 - val_loss: 4.0161
Epoch 43/50
563/563             5s 10ms/step -
accuracy: 0.8791 - loss: 0.3452 - val_accuracy: 0.4399 - val_loss: 4.0987
```

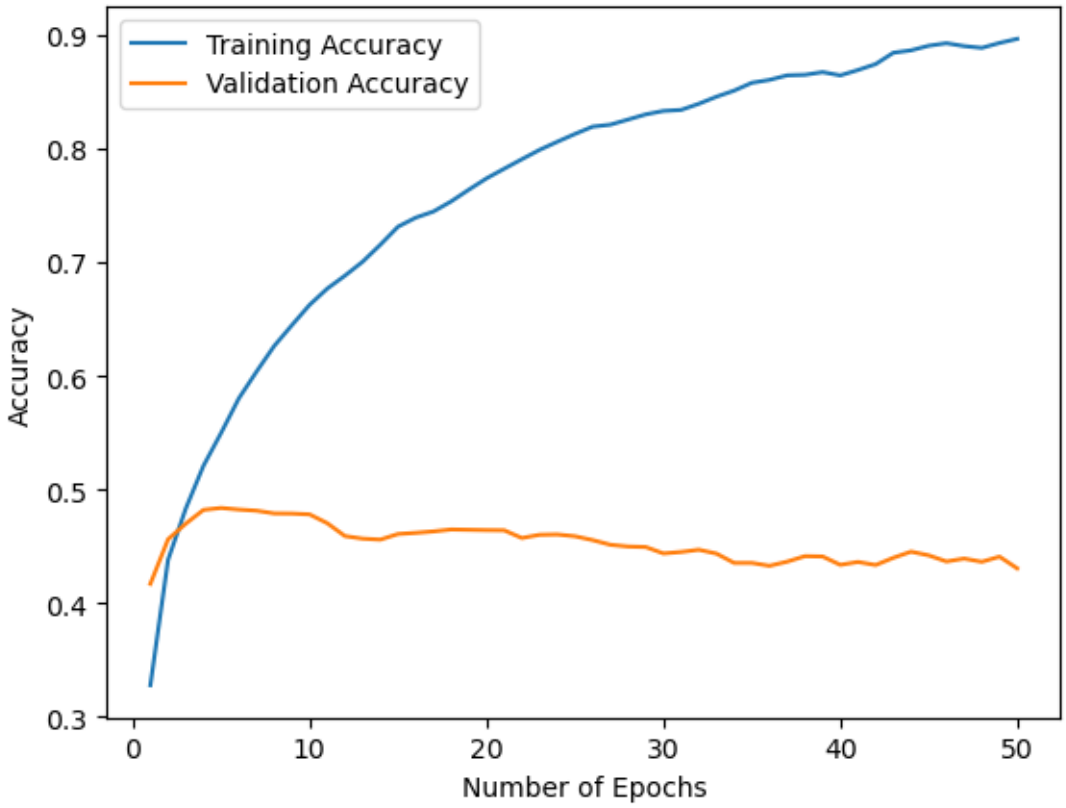
```
Epoch 44/50
563/563             4s 8ms/step -
accuracy: 0.8813 - loss: 0.3406 - val_accuracy: 0.4453 - val_loss: 4.1497
Epoch 45/50
563/563             5s 9ms/step -
accuracy: 0.8813 - loss: 0.3357 - val_accuracy: 0.4421 - val_loss: 4.2813
Epoch 46/50
563/563             5s 9ms/step -
accuracy: 0.8869 - loss: 0.3214 - val_accuracy: 0.4367 - val_loss: 4.3927
Epoch 47/50
563/563             5s 8ms/step -
accuracy: 0.8899 - loss: 0.3111 - val_accuracy: 0.4392 - val_loss: 4.4483
Epoch 48/50
563/563             4s 7ms/step -
accuracy: 0.8829 - loss: 0.3170 - val_accuracy: 0.4364 - val_loss: 4.5431
Epoch 49/50
563/563             5s 8ms/step -
accuracy: 0.8898 - loss: 0.3033 - val_accuracy: 0.4409 - val_loss: 4.6419
Epoch 50/50
563/563             5s 9ms/step -
accuracy: 0.8958 - loss: 0.2883 - val_accuracy: 0.4306 - val_loss: 4.7518
```

[234]:
```
training_accuracy = history.history['accuracy']
validation_accuracy = history.history['val_accuracy']
epoch_number = np.arange(1,51)
```

Here I plotted the training accuracy and validation accuracy against the number of epochs. The training accuracy continues to as the epochs rise, but the validation accuracy levels off after about 6 epochs.

[236]:
```
plt.plot(epoch_number, training_accuracy, label = 'Training Accuracy')
plt.xlabel("Number of Epochs")
plt.ylabel("Accuracy")
plt.plot(epoch_number, validation_accuracy, label = 'Validation Accuracy')
plt.legend()
```

[236]:
```
<matplotlib.legend.Legend at 0x1bdead13620>
```

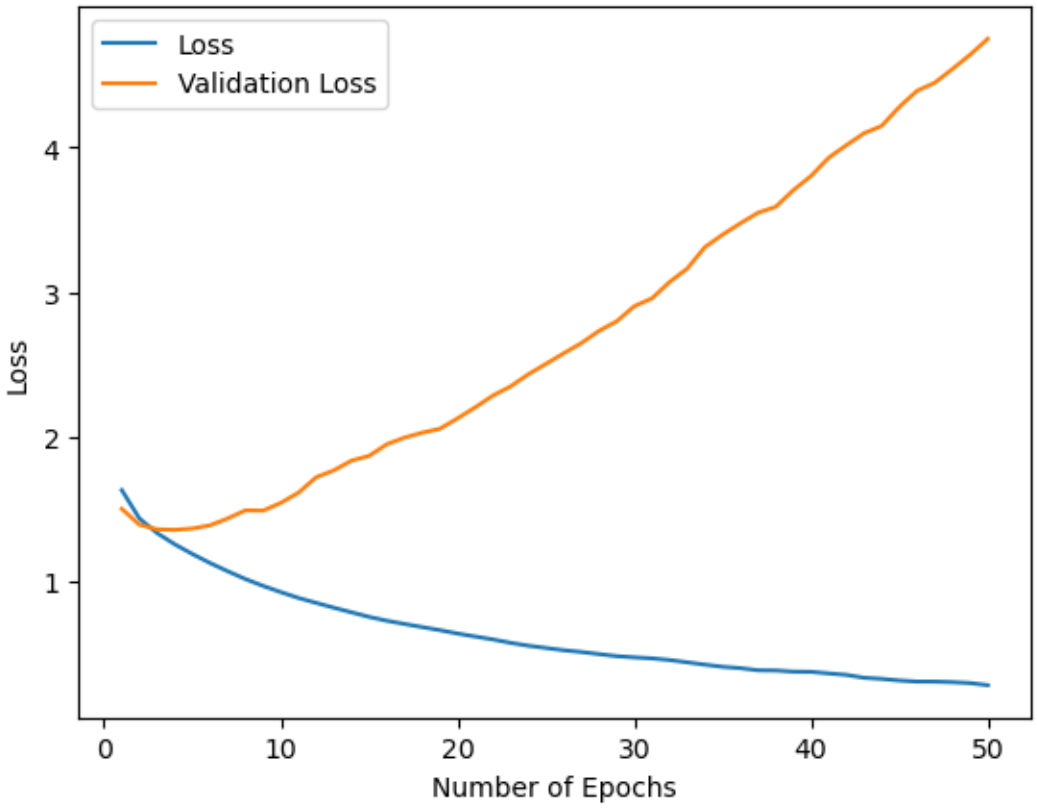


[237]:
```
loss = history.history['loss']
validation_loss = history.history['val_loss']
```

I plotted loss vs validation loss and found that to minimize the validation loss I want around 4 epochs. After 4 epochs the validation loss begins to increase. Due to this and the information about the accuracy, I chose 4 epochs for the fitting for my model.

[239]:
```
plt.plot(epoch_number, loss, label = 'Loss')
plt.xlabel("Number of Epochs")
plt.ylabel("Loss")
plt.plot(epoch_number, validation_loss, label = 'Validation Loss')
plt.legend()
```

[239]:
```
<matplotlib.legend.Legend at 0x1be4d305730>
```



Here I fit the model again with 4 epochs.

[241]:
```
np.random.seed(234) #set the seed in numpy
tf.random.set_seed(2345)  #set the seed in tensorflow



mynet = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48,␣
 ↪3)),
  tf.keras.layers.MaxPooling2D((2, 2)),
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(6, activation='softmax')
], name = 'mynet')
```

[242]:
```
mynet.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

[243]:
```
history_2 = mynet.fit(x=train_set_nn, y=train_set_labels,␣
↪validation_data=(test_set_nn, test_set_labels), epochs=4, batch_size=32)
```

```
Epoch 1/4
563/563             5s 8ms/step -
accuracy: 0.2606 - loss: 1.7240 - val_accuracy: 0.4239 - val_loss: 1.4971
Epoch 2/4
563/563             5s 8ms/step -
accuracy: 0.4193 - loss: 1.4687 - val_accuracy: 0.4636 - val_loss: 1.3757
Epoch 3/4
563/563             5s 10ms/step -
accuracy: 0.4714 - loss: 1.3625 - val_accuracy: 0.4780 - val_loss: 1.3524
Epoch 4/4
563/563             4s 7ms/step -
accuracy: 0.5024 - loss: 1.2928 - val_accuracy: 0.4890 - val_loss: 1.3382
```

Next, I calculated used my model to make predictions with the test data.

[245]:
```
predictions_nn = mynet.predict(test_set_nn)
```

```
221/221             0s 2ms/step
```

[246]:
```
predicted_labels_nn = np.zeros(len(predictions_nn))

for i in range(len(predictions_nn)):
    predicted_labels_nn[i] = np.argmax(predictions_nn[i])
```

Here I found that the accuracy for the test data predictions was about 49%, which is decent, but not ideal.

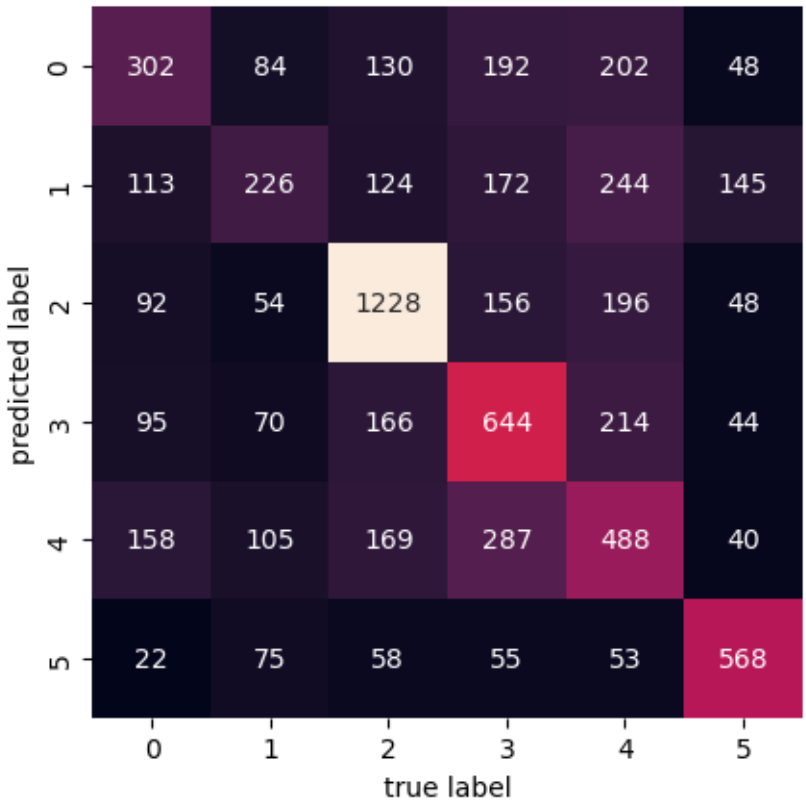
[248]:
```
test_loss, test_acc = mynet.evaluate(test_set_nn, test_set_labels, verbose=2)
print("\nEmotional Classification Validation Accuracy for Neural Net Model:",␣
↪test_acc)
```

```
221/221 - 0s - 2ms/step - accuracy: 0.4890 - loss: 1.3382

Emotional Classification Validation Accuracy for Neural Net Model:
0.4890335500240326
```

I created a confusion matrix using the predicted labels and the true labels for the test data.

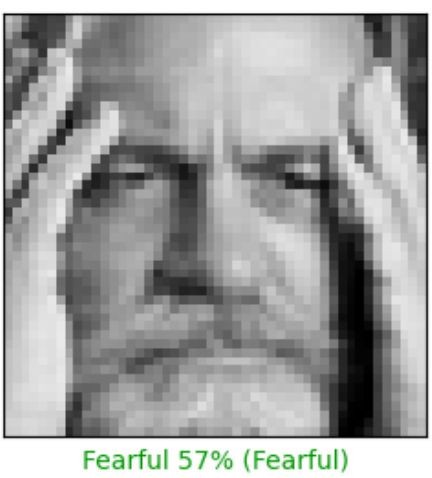
[250]:
```
mat_nn = confusion_matrix(predicted_labels_nn, test_set_labels)
sns.heatmap(mat_nn.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels= class_names_numbers,
            yticklabels= class_names_numbers)
plt.xlabel('true label')
plt.ylabel('predicted label');
```

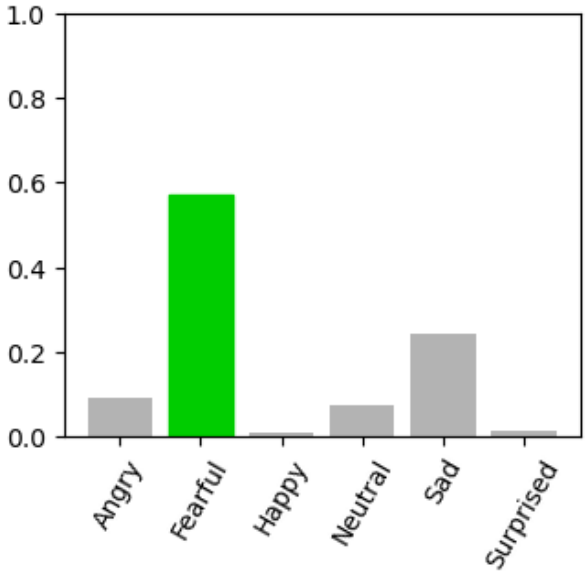


Testing the model on one of the images from the test set. It correctly guessed the expression of the image.

[252]:
```
i = 1330

plt.figure(figsize=(8,3))
plt.subplot(1,2,1)
plot_image(i, predictions_nn[i], test_set_labels, test_set)
plt.subplot(1,2,2)
plot_value_array(i, predictions_nn[i],  test_set_labels)
plt.xticks(range(6), class_names, rotation=60)
plt.yticks(0.2*np.arange(6))
plt.show()
```





Next up I used a Random Forest classification technique.

Random Forest:

Concepts From: https://www.geeksforgeeks.org/random-forest-for-image-classification-using-opencv/

I imported more important modules.

[257]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from skimage.feature import hog
import matplotlib.pyplot as plt
import os
import random
```

Extracting an image's hog features is a convenient way to simplify image data into easily usable chunks of data in order to simplify the creation of things like random forests. Here is a function that extracts the hog features.

[259]:
```
def extract_hog_features(image):
    # Calculate HOG features
    hog_features = hog(image, orientations=9, pixels_per_cell=(8, 8),
    cells_per_block=(2, 2), visualize=False, channel_axis = -1)
    return hog_features
```

This extracts hog features for input arrays.

[261]:
```
def load_and_extract_features(array_rf):
    X_rf = []
```

```
    for i in range(len(array_rf)):
        hog_features = extract_hog_features(array_rf[i])
        X_rf.append(hog_features)
    return X_rf
```

This function trains random forests.

[263]:
```
def train_random_forest(X_train_rf, y_train_rf):
    rf_classifier = RandomForestClassifier(n_estimators=1000, criterion='gini',␣
 ↪max_depth=5)
    rf_classifier.fit(X_train_rf, y_train_rf)
    return rf_classifier
```

Using the created function to extract the hog features of the training data.

[265]:
```
emotion_train_X_rf = load_and_extract_features(train_set)
emotion_train_y_rf = train_set_labels
```

Next, I used the above random forest function to train a random forest using the training data.

[267]:
```
emotion_rf_classifier = train_random_forest(emotion_train_X_rf,␣
 ↪emotion_train_y_rf)
```

I performed the same hog extraction technique on the test set.

[269]:
```
emotion_test_X_rf = load_and_extract_features(test_set)
emotion_test_y_rf = test_set_labels
```

I then used my random forest to make predictions based off the test data. The accuracy of the random forest predictions based off the test set was nearly 40%. This means it is likely more accurate for out of sample data than the neural net.

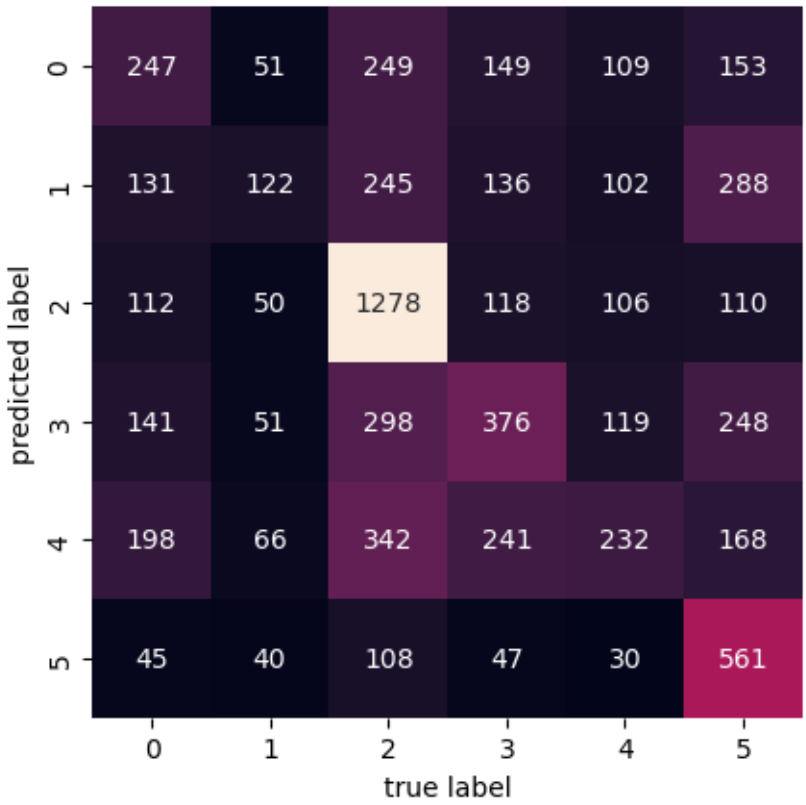
[271]:
```
emotion_predictions_rf = emotion_rf_classifier.predict(emotion_test_X_rf)
emotion_accuracy_rf = accuracy_score(emotion_test_y_rf, emotion_predictions_rf)

print(f"Emotion Classification Validation Accuracy for Random Forest Model:␣
 ↪{emotion_accuracy_rf}")
```

```
Emotion Classification Validation Accuracy for Random Forest Model:
0.3984717701995189
```

Another confusion matrix but for the random forest predictions this time.

[273]:
```
mat_rf = confusion_matrix(emotion_predictions_rf, test_set_labels)
sns.heatmap(mat_rf.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels= class_names_numbers,
            yticklabels= class_names_numbers)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



I chose another random image to test, and this model correctly predicted the emotion label.

[275]:
```
i = 3678
plt.figure(figsize=(8,3))
plt.subplot(1,2,1)
plot_image_2(i, emotion_predictions_rf, test_set_labels, test_set)
```

K Clustering:

Concepts From: https://medium.com/@joel_34096/k-means-clustering-for-image-classification-a648f28bdc47

K-means clustering can only be used on arrays of particular dimensions. I needed to make my arrays dim 2 or less. So, I reshaped them.

[279]:
```
train_set_kcl = train_set.reshape(len(train_set),-1)
test_set_kcl = test_set.reshape(len(test_set),-1)
```

[280]:
```
print(train_set_kcl.shape)
print(test_set_kcl.shape)
```

```
(18000, 6912)
(7067, 6912)
```

[281]:
```
from sklearn.cluster import MiniBatchKMeans
```

I created a minibatchkmeans model and fit it using the k clustering training set.

[283]:
```
total_clusters = len(np.unique(test_set_labels))

kmeans = MiniBatchKMeans(n_clusters = total_clusters, batch_size = 2048)
# Fitting the model to training set
kmeans.fit(train_set_kcl)
```

[283]:
```
MiniBatchKMeans(batch_size=2048, n_clusters=6)
```

[284]:
```
kmeans.labels_
```

[284]:
```
array([2, 1, 5, …, 3, 0, 0])
```

I calculated the accuracy for the predictions of the clustering model based off the testing data. It had an test set prediction accuracyy of 17% which is particularly bad compared to the first two models.

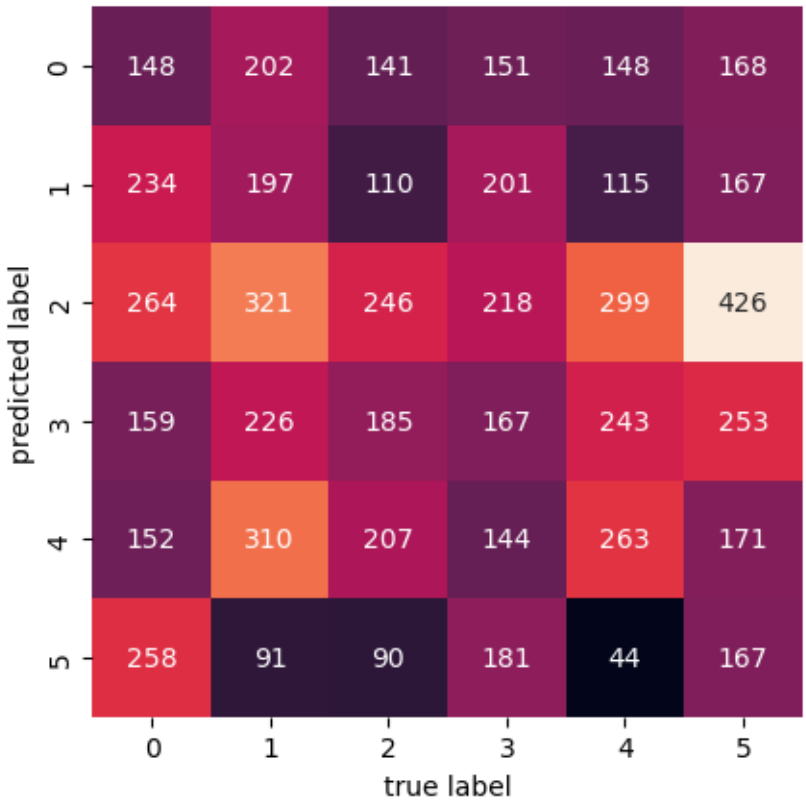
[286]:
```
emotion_predictions_kcl = kmeans.predict(test_set_kcl)
emotion_accuracy_kcl = accuracy_score(test_set_labels, emotion_predictions_kcl)
print(f"Emotion Classification Validation Accuracy For MiniBatchKmeans␣
 ↪Clustering Model: {emotion_accuracy_kcl}")
```

```
Emotion Classification Validation Accuracy For MiniBatchKmeans Clustering Model:
0.16810527805292202
```

A confusion matrix for the clustering model

[288]:
```
mat_kcl = confusion_matrix(emotion_predictions_kcl, test_set_labels)
sns.heatmap(mat_kcl.T, square=True, annot=True, fmt='d', cbar=False,
```

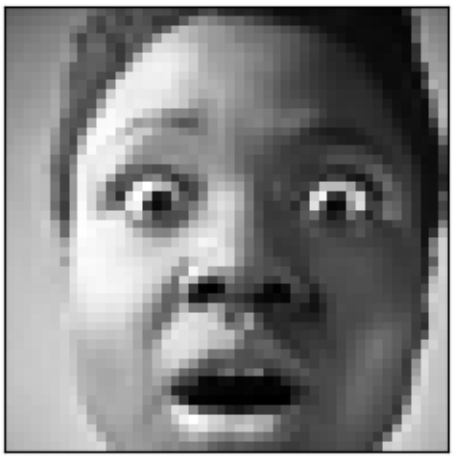
```
            xticklabels= class_names_numbers,
            yticklabels= class_names_numbers)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



Unsurprisingly, the model did not correctly guess the emotion of a randomly chosen image.

[290]:
```
i = 7002

plt.figure(figsize=(8,3))
plt.subplot(1,2,1)
plot_image_2(i, emotion_predictions_kcl, test_set_labels, test_set)
```



Conclusion:

The models varied greatly in their predictive abilities. The convolution neural net had an out of sample accuracy score of 49%. The clustering model had an out of sample accuracy score of 17%, and the random forest model had an out of sample accuracy score of 40%. The neural net was the best model in terms of out of sample accuracy. Because random forests are scalable, can measure relative importance of features, and are less prone to over-fitting than other tree types, I believe that the random forest might still has some benefits over the neural net. I think with different activation functions, it's possible that the neural net could perform even better. It's almost 10% more accurate on out of sample data than random forest, but I think the random forest is still useful for how powerful it while being so simple and without tweaking. The random forest and the neural net are much better at determining the correct emotions displayed by the images than they would be by pure chance, so the model has some use, but I think that more accuracy is needed in order to justify using this model to make accurate predictions for practical purposes.