Emotion Detection: a Machine Learning Project

A computer vision project about emotion detection

Aarohi Gupta Dec 28, 2019 · 6 min read ★

Emotion detection (n.):

The process of identifying human emotion

If someone showed you a picture of a person and asked you to guess what they're feeling, chances are you'd have a pretty good idea about it. What if your computer could do the same? What if it could become even better than you are? It seems like an absurd thought, right?

In my last blog (*read: Demystifying Artificial Intelligence*), I had said that I would explain a project that would show how the concepts discussed are applied.

The following blog contains a recollection of all of the series of things that we did during the project with InspiritAI, while it does show a lot of the code required to do Emotion Detection, this is by no means the fastest way to train a model. Multiple ML and AI models were used to see the difference between them.

The three main components of Emotion Detection are as follows:

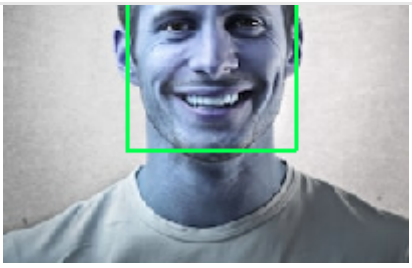
1. Image Preprocessing
2. Feature Extraction

Face detection:

Facial detection is an important step in emotion detection. It removes the parts of the image that aren't relevant. Here's one way of detecting faces in images.

```
import dlib
import numpy as np

frontalface_detector = dlib.get_frontal_face_detector()

def rect_to_bb(rect):
    x = rect.left()
    y = rect.top()
    w = rect.right() - x
    h = rect.bottom() - y
    return (x, y, w, h)

def detect_face(image_url):
    try:
        url_response = urllib.request.urlopen(image_url)
        img_array = np.array(bytearray(url_response.read()),
dtype=np.uint8)
        image = cv2.imdecode(img_array, -1)

rects = frontalface_detector(image, 1)

if len(rects) < 1:
    return "No Face Detected"

for (i, rect) in enumerate(rects):
    (x, y, w, h) = rect_to_bb(rect)
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

plt.imshow(image, interpolation='nearest')
plt.axis('off')
plt.show()
```

Another way to do this is by using dlib's pre-trained face detector model which is used in the next point as well.



Facial landmarks:

Facial landmarks are a set of key points on human face images. The points are defined by their (x,y) coordinates on the image. These points are used to locate and represent salient regions of the face, such as eyes, eyebrows, nose, mouth and jawline.

The facial landmark model we used was the Dlib's pre-trained Facial Landmark Detection Model which detects 68 2-Dimensional points on the human face.

You can load the model like this:

```
import dlib
import numpy as np

frontalface_detector = dlib.get_frontal_face_detector()

landmark_predictor=dlib.shape_predictor('./shape_predictor_68_face_l
andmarks.dat')

def get_landmarks(image_url):
    try:
        url_response = urllib.request.urlopen(image_url)
        img_array = np.array(bytearray(url_response.read()),
dtype=np.uint8)
        image = cv2.imdecode(img_array, -1)

    except Exception as e:
        print ("Please check the URL and try again!")
        return None,None
    faces = frontalface_detector(image, 1)

    if len(faces):
        landmarks = [(p.x, p.y) for p in landmark_predictor(image,
faces[0]).parts()]

    else:
        return None,None

    return image,landmarks
```

```
    image_copy = image.copy()
    for (x, y) in face_landmarks:
        cv2.circle(image_copy, (x, y), circle_thickness, (255,0,0),
radius)
        plt.imshow(image_copy, interpolation='nearest')
        plt.axis('off')
        plt.show()
```

After using the model, your output should look like this:

In this model, the specific landmarks for facial features are:

```
Jawline = 0-16
Eyebrows = 17-26
Nose = 27-35
Left eye = 36-41
Right eye = 42-47
Mouth = 48-67
```

One way to differentiate between two emotions is to see whether the persons mouth and eyes are open or not. We can find the euclidian distance between the points specifically on the mouth, if a person is surprised, the distance would be more than it would be if they're not.

Data Pre-processing

Before using the data, it is important to go through a series of steps called pre-processing. This makes the data easier to handle.

We will use a modified version of the fer2013 dataset consisting of five emotion labels.

- Pixels of the image stored in string format
- Integer encoding of the target label

There is a total of 20,000 images distributed equally across the five emotions. The images are 48*48 grayscale cropped images. The csvfile consists of a flattened array of the image stored in the form of a string

```
The target labels are integer encoded in the csvfile. They are
mapped as follows:

0 — -> Angry
1 — -> Happy
2 — -> Sad
3 — -> Surprise
4 — -> Neutral
```

Load the dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

label_map=
{"0":"ANGRY","1":"HAPPY","2":"SAD","3":"SURPRISE","4":"NEUTRAL"}

df = pd.read_csv("./ferdata.csv")

df.head()
```

This dataset contains the raw pixel values of the images.

Split the data

As discussed last time, the data needs to be split into two different sets:

1. Training set: the algorithm will read, or ‘train’, on this over and over again to try and learn its task.
2. Testing set: the algorithm is tested on on this data to see how well it works.

```
X_train, X_test, y_train, y_test = train_test_split(dataX, dataY,
test_size=0.1,random_state=42,stratify =dataY)
```

Standardise the data

Standardization is the process of putting different variables on the same scale. It rescales data to have a mean of 0 and a standard deviation of 1. This transformation centres the data.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Linear models

K-nearest neighbours

KNN is a non-parametric learning algorithm, meaning it does not make any assumptions about the distribution of data. We used the euclidean distances between the points as the data.

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, Y_train)
predictions = knn.predict(X_test)
```

To find the accuracy of the model:

```
from sklearn.metrics import accuracy_score

print(accuracy_score(Y_test, predictions)*100)
```

Our accuracy was about 50% so we tried some non-linear models.

Non-linear models

Multi-layer perceptrons

MLPs are a subcategory of neural networks. They are made of one or more layers of neurons. The input layers are where the data is fed, after which there may be one or more hidden layers. The predictions come from the output layer.

```
from keras.models import Sequential
from keras.utils import to_categorical
from keras.layers import Dense, Dropout, Flatten, Activation
from keras.losses import categorical_crossentropy
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.models import load_model
from keras.optimizers import Adam, SGD

mlp_model = Sequential()

mlp_model.add(Dense(1024, input_shape = (2304,), activation =
'relu', kernel_initializer='glorot_normal'))

mlp_model.add(Dense(512, activation = 'relu',
kernel_initializer='glorot_normal'))

mlp_model.add(Dense(5, activation = 'softmax'))

mlp_model.compile(loss=categorical_crossentropy,
optimizer=SGD(lr=0.001), metrics=['accuracy'])

checkpoint = ModelCheckpoint('best_mlp_model.h5',verbose=1,
monitor='val_acc', save_best_only=True,mode='auto')

mlp_history = mlp_model.fit(X_train, y_train, batch_size=batch_size,
epochs=epochs, verbose=1, callbacks=[checkpoint],
validation_data(X_test, y_test),shuffle=True)
```

Our accuracy using pixels was about 50%, this increased when instead of using pixel values, we used the distances between the facial landmarks. However we wanted models that would be even more accurate, so we decided to use CNNs.

Convolutional Neural Networks

```
width, height = 48, 48
```

```
X_train = np.expand_dims(X_train,3)

X_test = np.expand_dims(X_test,3)

cnn_model = Sequential()

cnn_model.add(Conv2D(5000, kernel_size=(4, 4), activation='relu',
padding='same', input_shape = (width, height, 1)))

cnn_model.add(BatchNormalization())

cnn_model.add(MaxPooling2D(pool_size=(3, 3), strides=(4, 4)))

cnn_model.add(Dropout(0.2))

cnn_model.add(Flatten())

cnn_model.add(Dense(2000, activation='relu'))

cnn_model.add(Dropout(0.2))

cnn_model.add(Dense(5, activation='softmax'))

checkpoint = ModelCheckpoint('best_cnn_model.h5', verbose=1,
monitor='val_loss',save_best_only=True, mode='auto')

cnn_model.compile(loss=categorical_crossentropy,
optimizer=Adam(lr=0.001, beta_1=0.9, beta_2=0.999),
metrics=['accuracy'])

cnn_history = cnn_model.fit(X_train, y_train, batch_size=batch_size,
epochs=epochs, verbose=1, callbacks=[checkpoint],
validation_data=(X_test, y_test),shuffle=True)

cnn_model.save('cnn_model.h5')
```

To improve performance you can very the dropout, number of dense layers and activation functions. We also used transfer learning with a CNN called VGG which is a pre-trained convolutional neural network for image classification.

Evaluation

The best results came by using the VGG which were right about 68–70% of the time, but even the linear models did a very good job. Although 50% accuracy does not

The VGG, however, performs even better than humans do in this particular dataset. The difference between the CNNs and the MLPs was that the CNNs were extracting features that they deemed important by themselves, while we were feeding either pixels or landmarks as features to the MLPs.

For more information on KNNs, CNNs, MLPs and other basic machine learning topics click on this link.

Special thanks to Adeesh Goel for organising this amazing workshop, check out the facebook page of InspiritAI, Tyler Bonnen for being such a wonderful mentor and guiding us through the program, and my team members Khushi and Harsh.

Machine Learning Emotion Detection Computer Vision Artificial Intelligence Computer Science