Facial Expression Recognition

Author: Sahil Bharodiya
sahilbharodiya2002@outlook.com
Dated: 16th May 2021

Abstract

This is a Convolutional Neural Network based model (read more about CNN). I used python tools and libraries like OpenCV for image related work, NumPy for array and list related work, Matplotlib for plotting image and curves, Sci-Kit learn for dividing datasets, TensorFlow for training, testing and evaluating Neural Network. For learning purposes comments and heading are given in google colab notebook (Google Colab is a free jupyter-like notebook for python coding and you don't need to install any libraries in your local machine which are mentioned above, just import and code). In the end I saved the model in .hf extension and also provided a code for working with a pretrained model. **For best experience try to run this code in google colab.** In the reference section at last I provided a ScienceDirect article link for knowing mathematics behind Facial Expression Recognition.

About Data

Data used in this complete code is facial expression data downloaded from kaggle. Total seven types of expressions it has. Folders are named as '0'–Angry, '1'–Disgust, '2'–Fear, '3'–Happy, '4'–Sad, '5'–Surprise and '6'–Neutral. All images are of dimension 48 ×48 ×3.

About Process

First I converted all the image's pixels in size ranging from 0 to 1 which previously was 0 to 255 just for simplicity. Then I appended all image arrays in a list 'X' and simultaneously appended their expression code (mentioned in About Data) in the label list 'y'. Now, I converted this X and y into numpy arrays. Then I divided this dataset into training, testing and validating using train_test_split. Now, our 60% of data is for training, 20% for testing and 20% for validating.

About CNN that is trained in this model

The 8 lines of code below define the convolutional base using a common pattern: a stack of Conv2D and MaxPooling2D layers.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
```

As input, a CNN takes tensors of shape (image_height, image_width, color_channels), ignoring the batch size. If you are new to these dimensions, color_channels refers to (R, G, B). In this example, you will configure our CNN to process inputs of shape (48, 48, 3), which is the format of these images. You can do this by passing the argument input_shape to our first layer. Above, you can see that the output of every Conv2D and MaxPooling2D layer is a 3D tensor of shape (height, width, channels). The width and height dimensions tend to shrink as you go deeper in the network. The number of output channels for each Conv2D layer is controlled by the first argument (e.g., 32 ,64 or 128). Typically, as the width and height shrink, you can

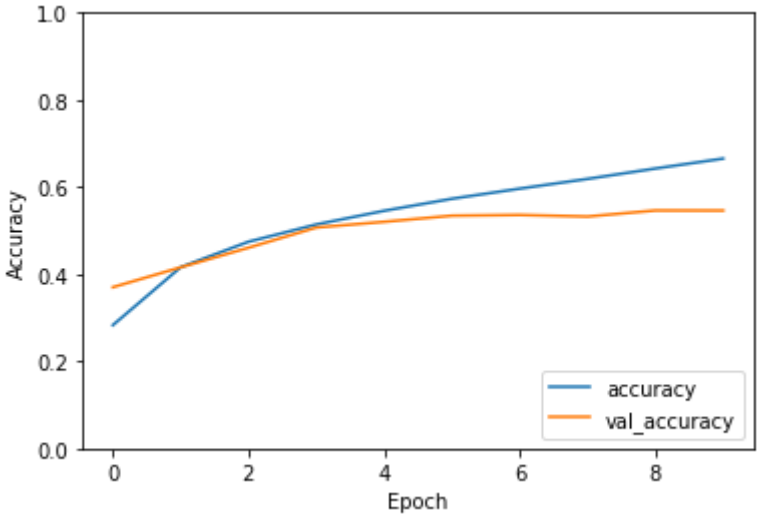
afford (computationally) to add more output channels in each Conv2D layer.

Add Dense layers on top: To complete our model, you will feed the last output tensor from the convolutional base (of shape (2, 2, 128)) into one or more Dense layers to perform classification. Dense layers take vectors as input (which are 1D), while the current output is a 3D tensor. First, you will flatten (or unroll) the 3D output to 1D, then add one or more Dense layers on top. Facial Expression Image data has 7 output classes, so you use a final Dense layer with 7 outputs. Here is model summary:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 46, 46, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 21, 21, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 64)          36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 2, 2, 128)         73856
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 64)                32832
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 455
=================================================================
Total params: 167,623
Trainable params: 167,623
Non-trainable params: 0
_________________________________________________________________
```

Model Accuracy and Learning Curve

When I evaluated the model accuracy was 54.61%. Here is 'Accuracy' vs 'ephoches' graph. You can modify the ephoches.



Saving Model

Save model in .h5 format. Then just provide a link of .h5 saved model and run the code in a new or existing notebook. Then give a link of any facial image of any size. The code will prepare your image and predict the expression.

*** Happy Learning ***

References

To know mathematics behind facial expression recognition read Mathematical Representations of Blended Facial Expressions towards Facial Expression Modeling