Facial Expression Recognition

Using Deep Neural Networks

Muhammad Tajwar

Cafoscari University of Venice

June 28, 2021

Outline

Introduction to Keras

Deep Learning For Humans

- Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load, it offers consistent and simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear actionable error messages. It also has extensive documentation and developer guides.
- Keras is the most used deep learning framework among top-5 winning teams on Kaggle. Because Keras makes it easier to run new experiments, it empowers you to try more ideas than your competition, faster. And this is how you win.
- Built on top of TensorFlow 2.0, Keras is an industry-strength framework that can scale to large clusters of GPUs or an entire TPU pod.

Problem Introduction

- Facial expression recognition plays an important role in communicating the emotions and intentions of human beings. Facial expression recognition in uncontrolled environment is more difficult as compared to that in controlled environment due to change in occlusion, illumination, and noise.
- In this Project, I present a framework for effective facial expression recognition from real-time facial images.
- This method extracts the discriminative feature from salient face regions and then combine with texture and orientation features for better representation.
- The proposed framework is capable of providing high recognition accuracy rate even in the presence of occlusions, illumination, and noise.
- I use The CNN Convolutional Neural Network so solve this problem.

The Dataset

- I use 5 facial expressions for the learning of the neural network
- Mainly Happy,Sad,Angry,Surprise and Neural
- Each of expression folder has almost 5000 to 7000 images
- For the training and the validation i choose 70:30

CNN

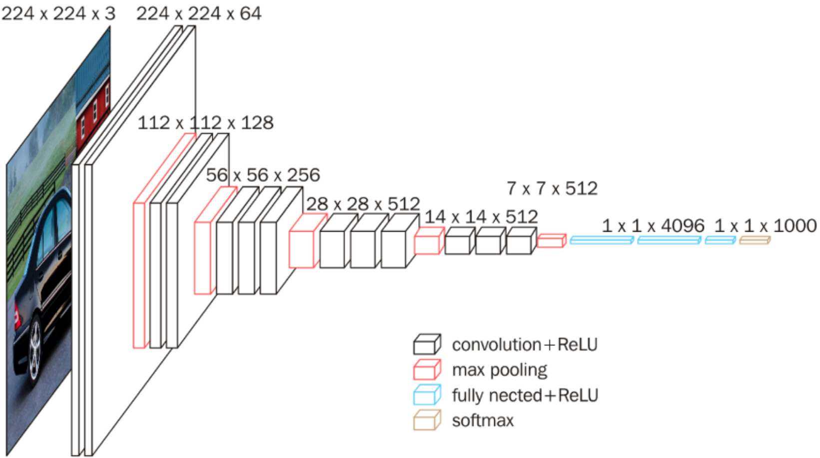
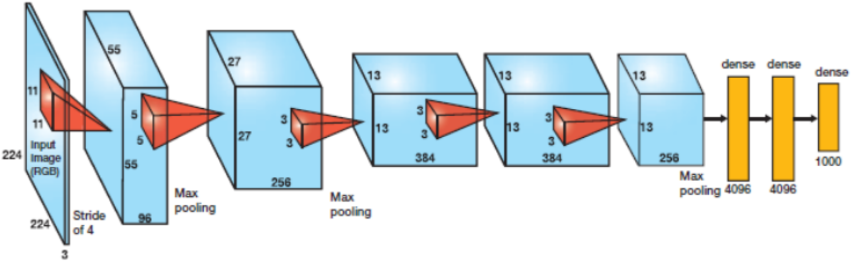


Figure: CNN Layers

AlexNet



AlexNet Model Architecture

Block 1

```
#Block 1
#Layer 1
model.add(Conv2D(32, #We are using 32 neurons
            (3,3),
            padding='same',
            kernel_initializer='he_normal',
            input_shape=(img_rows,img_cols,1))
            )
model.add(Activation('elu'))
model.add(BatchNormalization())
#Layer 2
model.add(Conv2D(32,
            (3,3),
            padding='same',
            kernel_initializer='he_normal',
            input_shape=(img_rows,img_cols,1))
            )
model.add(Activation('elu'))
model.add(BatchNormalization())
#layer 3 Maxpooling Layer
model.add(MaxPooling2D(pool_size=(2,2)))
#Layer 4 Dropout Layer
model.add(Dropout(0.2))
```

Block 2

```
#Block 2
#Layer 1
model.add(Conv2D(64, #We are using 64 neurons
          (3,3),
          padding='same',
          kernel_initializer='he_normal')
          )
model.add(Activation('elu'))
model.add(BatchNormalization())
#Layer 2
model.add(Conv2D(64,
          (3,3),
          padding='same',
          kernel_initializer='he_normal')
          )
model.add(Activation('elu'))
model.add(BatchNormalization())
#layer 3 Maxpooling Layer
model.add(MaxPooling2D(pool_size=(2,2)))
#Layer 4 Dropout Layer
model.add(Dropout(0.2))
```

Block 3

```
#Block 3
#Layer 1
model.add(Conv2D(128, #We are using 128 neurons
                 (3,3),
                 padding='same',
                 kernel_initializer='he_normal')
                 )
model.add(Activation('elu'))
model.add(BatchNormalization())
#Layer 2
model.add(Conv2D(128,
                 (3,3),
                 padding='same',
                 kernel_initializer='he_normal')
                 )
model.add(Activation('elu'))
model.add(BatchNormalization())
#layer 3 Maxpooling Layer
model.add(MaxPooling2D(pool_size=(2,2)))
#Layer 4 Dropout Layer
model.add(Dropout(0.2))
```

Block 4

```
#Block 4
#Layer 1
model.add(Conv2D(256, #We are using 256 neurons
          (3,3),
          padding='same',
          kernel_initializer='he_normal')
          )
model.add(Activation('elu'))
model.add(BatchNormalization())
#Layer 2
model.add(Conv2D(256,
          (3,3),
          padding='same',
          kernel_initializer='he_normal')
          )
model.add(Activation('elu'))
model.add(BatchNormalization())
#layer 3 Maxpooling Layer
model.add(MaxPooling2D(pool_size=(2,2)))
#Layer 4 Dropout Layer
model.add(Dropout(0.2))
```

Block 5

```
#Block 5
#Layer 1
#Flattening layer
model.add(Flatten())
model.add(Dense(64,
                kernel_initializer='he_normal')
                )
model.add(Activation('elu'))
model.add(BatchNormalization())
#Layer 2 Dropout Layer
model.add(Dropout(0.5))
```

Block 6

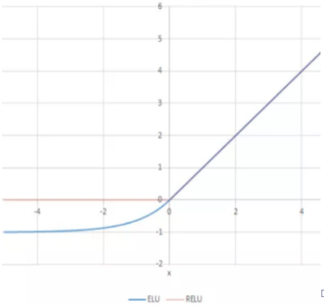
```
#Block 6
#Layer 1
model.add(Dense(64,
                kernel_initializer='he_normal')
                )
model.add(Activation('elu'))
model.add(BatchNormalization())
#Layer 2 Dropout Layer
model.add(Dropout(0.5))
```

Block 7

```
#Block 7 Last Output Layer
#Layer 1
model.add(Dense(num_classes,
                kernel_initializer='he_normal')
                )
model.add(Activation('softmax'))
```

ELU

- Exponential Linear Unit or its widely known name ELU is a function that tend to converge cost to zero faster and produce more accurate results. Different to other activation functions, ELU has a extra alpha constant which should be positive number.
- ELU is very similiar to RELU except negative inputs. They are both in identity function form for non-negative inputs. On the other hand, ELU becomes smooth slowly until its output equal to $-\infty$ whereas RELU sharply smoothes



Learning

```
Epoch 1/25
755/755 [==============================] - 531s 701ms/step - loss: 2.1126 - accuracy: 0.2189 -

Epoch 00001: val_loss improved from inf to 1.52976, saving model to Emotion_little_vgg.h5
Epoch 2/25
755/755 [==============================] - 531s 704ms/step - loss: 1.5886 - accuracy: 0.2734 -

Epoch 00002: val_loss improved from 1.52976 to 1.51209, saving model to Emotion_little_vgg.h5
Epoch 3/25
755/755 [==============================] - 530s 702ms/step - loss: 1.5505 - accuracy: 0.2951 -

Epoch 00003: val_loss did not improve from 1.51209
Epoch 4/25
755/755 [==============================] - 530s 702ms/step - loss: 1.5303 - accuracy: 0.3086 -

Epoch 00004: val_loss improved from 1.51209 to 1.38252, saving model to Emotion_little_vgg.h5
Epoch 5/25
755/755 [==============================] - 531s 703ms/step - loss: 1.4752 - accuracy: 0.3497 -

Epoch 00005: val_loss improved from 1.38252 to 1.37406, saving model to Emotion_little_vgg.h5
Epoch 6/25
599/755 [=====================>.......] - ETA: 1:46 - loss: 1.4198 - accuracy: 0.3792
```

Learning

```
Epoch 00006: val_loss improved from 1.37406 to 1.07032, saving model to Emotion_little_vgg.h5
Epoch 7/25
755/755 [==============================] - 532s 704ms/step - loss: 1.3112 - accuracy: 0.4534

Epoch 00007: val_loss did not improve from 1.07032
Epoch 8/25
755/755 [==============================] - 530s 702ms/step - loss: 1.2445 - accuracy: 0.4931

Epoch 00008: val_loss improved from 1.07032 to 0.93466, saving model to Emotion_little_vgg.h5
Epoch 9/25
755/755 [==============================] - 530s 702ms/step - loss: 1.1978 - accuracy: 0.5052

Epoch 00009: val_loss did not improve from 0.93466
Epoch 10/25
755/755 [==============================] - 530s 702ms/step - loss: 1.1704 - accuracy: 0.5288

Epoch 00010: val_loss did not improve from 0.93466
Epoch 11/25
755/755 [==============================] - 531s 704ms/step - loss: 1.1351 - accuracy: 0.5429

Epoch 00011: val_loss improved from 0.93466 to 0.92017, saving model to Emotion_little_vgg.h5
Epoch 12/25
755/755 [==============================] - 562s 744ms/step - loss: 1.1116 - accuracy: 0.5586
```

Learning

```
Epoch 17/25
755/755 [==============================] - 537s 711ms/step - loss: 1.0549 - accuracy: 0.597

Epoch 00017: val_loss improved from 0.82345 to 0.79786, saving model to Emotion_little_vgg.
Epoch 18/25
755/755 [==============================] - 537s 711ms/step - loss: 1.0496 - accuracy: 0.588

Epoch 00018: val_loss did not improve from 0.79786
Epoch 19/25
755/755 [==============================] - 537s 712ms/step - loss: 1.0312 - accuracy: 0.597

Epoch 00019: val_loss did not improve from 0.79786
Epoch 20/25
755/755 [==============================] - 538s 712ms/step - loss: 1.0230 - accuracy: 0.602
Restoring model weights from the end of the best epoch.

Epoch 00020: val_loss did not improve from 0.79786

Epoch 00020: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
Epoch 00020: early stopping
```

Working Project