MACHINE LEARNING ENGINEER NANODEGREE

Capstone Project

Namita Atri
August 26th, 2018

I. DEFINITION

Project Overview

Interpersonal communication is an important part of every human's life, and technology nowadays is focused around it, let it be in the field of virtual reality, computer vision, or more broadly artificial intelligence. We interact not just by the method of verbal languages, but our non-verbal communication, such as emotions and expressions also form a vital part in communication. Owing to this, facial expression recognition is significantly being researched due to its large role in human interaction. Its applications are found in varied areas such as marketing, psychology, medical applications, robotics and so on.

To this end, I present a solution to implement **facial expression recognition** (here on referred to as FER), using deep learning methods. A part of this project, I will be working on the dataset obtained from a Kaggle competition. This dataset contains images (in form of pixels) representing a human face conveying an emotion which is then stored in the dataset as well and is denoted by a number (0-6) representing 7 basic emotions.

Problem Statement

The purpose of this project is to recognize and predict facial expression of a human when presented with his/her image, targeting the highest accuracy achievable keeping in mind limitations of the hardware being used for this task, time constraints, and the scope of the project's requirements.

I am using a deep learning approach using convolutional neural networks to solve the problem with the dataset 'fer2013' from the ICML 2013 challenge on Kaggle. This set was created as part of the ICML challenge in 2013 and contains 35887 entries of images denoted by pixels with corresponding values of emotion and it's intended usage.

The dataset presents seven expressions in total: Angry, Disgusted, Fearful, Happy, Sad, Surprised, Neutral. The usage will come into play while training the deep learning model and segregating the data int parts meant for training, testing etc.

The image size given is 48x48, and the dataset contains 28709 images in the training set, 3589 images in validation (labelled PublicTest) and testing (labelled PrivateTest) set each.

Solution Statement

I have attempted to solve the problem by means of deep learning, more specifically convolutional neural networks of six convolutional layers, and as part of refinement, varying parameters of the CNN. I will explain the models used in more detail at a later stage.

Let me first broadly explain the steps I have taken to get to the solution.

1. **Understanding the dataset**, its columns and what purpose each column serves. I recognize that I need to ensure that the data is in grayscale and is uniform (all images are the same size) to better the accuracy and also improve processing speed.
2. **Preprocessing the data**: I parsed each row, to retrieve the three components of the image, the emotion, the pixels, and its usage. I use this information to segregate the data into three parts for the purpose of storage- Training, PublicTest and PrivateTest. Within these, I store the images under respective category of the emotion. I have also created a file with the image serial number and value of the emotion.
   Process of normalization of numpy matrices converted from pixels and one-hot encoding on the emotion value is followed, to feed to the CNN.
3. I have **implemented a CNN model** then, with six layers of convolutional and a number of layers of max pooling, dense, batch normalization and dropouts, by varying hyperparameters, trained it on the training data and the validation data and then am using test data to find out the accuracy and the log loss.
4. After this, I repeat the process by **refining the model** by varying parameters, and training longer.
5. At the end, I **predict the emotion** on some new images that I provide which are external to the dataset.

Metrics

The accuracy is the ratio of true outcomes, which includes both true positives and true negatives, to the total number of cases examined (true outcomes and false outcomes).

Accuracy (ACC) = (TP + TN) / (TP + TN + FP + FN)

In the context of this project, accuracy can be reframed as the ratio of correct predictions to all predictions.

Accuracy (ACC) = Correct Predictions / All Predictions

The number of correct and total predictions can be easily retrieved while testing on the test data set.

II. ANALYSIS

Data Exploration

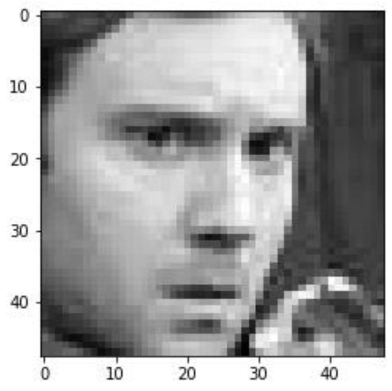
The dataset obtained is the FER2013 from the Kaggle challenge by ICML in 2013 for facial expression recognition. The data has 35887 images in total of 48x48 pixels each, all centred around the human face, with three components to every entry in the dataset, first for the emotion related to the image, second for the pixels of the image and the third defining its usage at the time of training the model. There are 7 basic emotions that are used in the dataset, and are as follows:

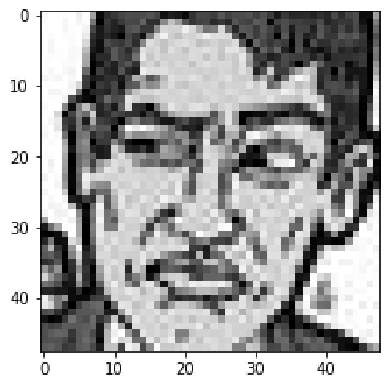
0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral

The three categories of usage are:
Training – for the purpose of training the model (28709 images)
PublicTest – for cross validation (3589 images)
PrivateTest – for testing the model (3589 images)

Emotion is one-hot encoded as this is going to be responsible for calculating accuracy during training.

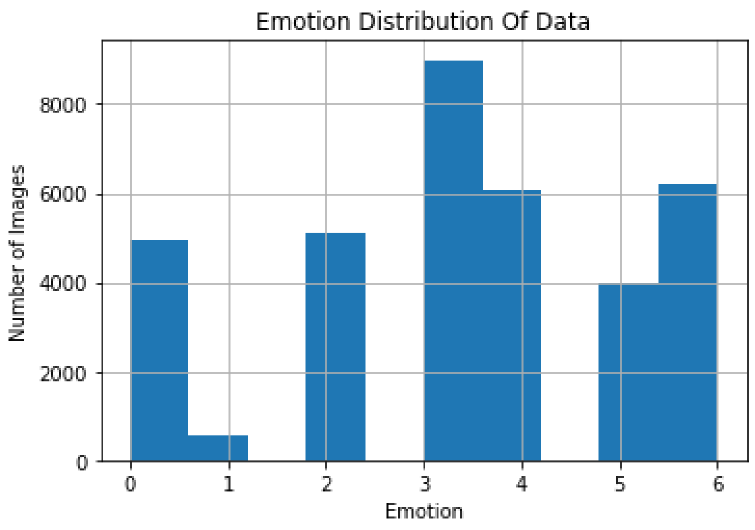
For exploration purpose, I show the images and the corresponding emotion value as well.
e.g., following expressions are 'Neutral'.





Exploratory Visualization

The images are distributed against emotions as per the following graph:



The counts are:

Happy	8989
Neutral	6198
Sad	6077
Fear	5121
Angry	4953
Surprise	4002
Disgust	547

We can easily see from the above visualization, the distribution of the dataset, with happy faces having the maximum weightage, and disgusted faces the least.

Algorithms and Techniques

(I have quoted the author of this Image Classification With CNN in this section)

Image classification is essentially taking an input and attaching a definition or class to it. This is a skill that humans learn from birth and would be easily able to tell that the image is that of an elephant. But the computer sees the pictures quite differently:

What I see
What a computer sees

Instead of the image, the computer sees an array of pixels. For example, if image size is 300 x 300. In this case, the size of the array will be 300x300x3. Where 300 is width, next 300 is height and 3 is RGB channel values.

To solve this problem the computer looks for the characteristics of the base level. In human understanding such characteristics are for example the trunk or large ears. For the computer, these characteristics are boundaries or curvatures. And then through the groups of convolutional layers the computer constructs more abstract concepts.

In more detail: the image is passed through a series of convolutional, nonlinear, pooling layers and fully connected layers, and then generates the output.

A CNN model starts with a **convolutional layer**. The image, which is a matrix with pixel values, is entered into it. Imagine that the reading of the input matrix begins at the top left of image. Next, the software selects a smaller matrix there, which is called a filter (or neuron, or core). Then the filter produces convolution, i.e. moves along the input image. The filter's task is to multiply its values by the original pixel values. All these multiplications are summed up. One number is obtained in the end. Since the filter has read the image only in the upper left corner, it moves further and further right by 1 unit performing a similar operation. After passing the filter across all positions, a matrix is obtained which is smaller than the matrix.

The network will consist of several convolutional networks mixed with nonlinear and pooling layers. When the image passes through one convolution layer, the output of the first layer becomes the input for the second layer. And this happens with every further convolutional layer.

The **nonlinear layer** is added after each convolution operation. It has an activation function, which brings nonlinear property. Without this property a network would not be sufficiently intense and will not be able to model the response variable (as a class label).

The **pooling layer** follows the nonlinear layer. It works with width and height of the image and performs a down-sampling operation on them. As a result the image volume is reduced. This means that if some features (as for example boundaries) have already been identified in the previous convolution operation, than a detailed image is no longer needed for further processing, and it is compressed to less detailed pictures.

After completion of series of convolutional, nonlinear and pooling layers, it is necessary to attach a **fully connected layer**. This layer takes the output information from convolutional networks. Attaching a fully connected layer to the end of the network results in an N dimensional vector, where N is the amount of classes from which the model selects the desired class.

Benchmark Model

The benchmark model I am using for this problem is a simple CNN model created from scratch, with the following architecture:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_66 (Conv2D)           (None, 48, 48, 4)         20        
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 24, 24, 4)         0         
_________________________________________________________________
conv2d_67 (Conv2D)           (None, 24, 24, 8)         136       
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 12, 12, 8)         0         
_________________________________________________________________
conv2d_68 (Conv2D)           (None, 12, 12, 16)        528       
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 6, 6, 16)          0         
_________________________________________________________________
flatten_12 (Flatten)         (None, 576)               0         
_________________________________________________________________
dense_41 (Dense)             (None, 7)                 4039      
=================================================================
Total params: 4,723.0
Trainable params: 4,723.0
Non-trainable params: 0.0
_________________________________________________________________
```

The test accuracy produced by this is 41%.

I am also comparing my implementation with that of Charlie Tang, who is the winner of a Kaggle challenge of facial expression recognition, and achieved an accuracy of 71.161%. I have tried reaching an accuracy as high as possible, but due to hardware limitations, my implementation results in lower accuracy. Charlie Tang used CNN in combination with SVMs, however I am using only CNN.

III. METHODOLOGY

Data Preprocessing

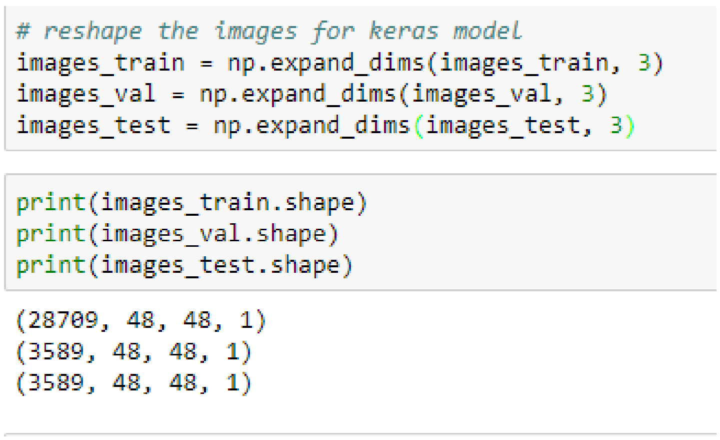
1. First, I read each row from fer2013.csv and parse it into three values – emotion, pixels and usage. I store values of emotions in order in emotions.csv, convert pixels in images and store them according to usage, Training, PublicTest and PrivateTest and the further sorted and stored in folders, the name of whose are values of emotion. The images are saved by their serial number in jpg format.

2. Numpy Arrays, images_train, images_val, images_test are created and stored on disk after dividing pixels by 255.

3. The output labels (value of emotion for each image) is fetched into y_train_encode, y_val_encode, y_test_encode for the next step.

4. I then pick the first image from every set and verify the label visually.

The image needs to be reshaped now to feed to the Keras model, and hence I use the following code to convert from 2-dimensional data to 4-dimensional data.



```
# reshape the images for keras model
images_train = np.expand_dims(images_train, 3)
images_val = np.expand_dims(images_val, 3)
images_test = np.expand_dims(images_test, 3)
```

```
print(images_train.shape)
print(images_val.shape)
print(images_test.shape)
```

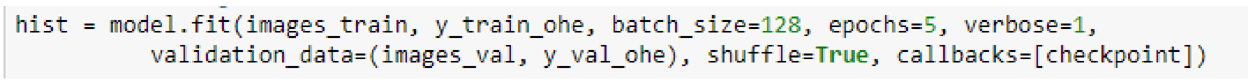
```
(28709, 48, 48, 1)
(3589, 48, 48, 1)
(3589, 48, 48, 1)
```

The last step is to verify that the shape of the data is as expected, as seen in the above image.

Implementation

Methodology to arrive at a solution comprises of the below steps:

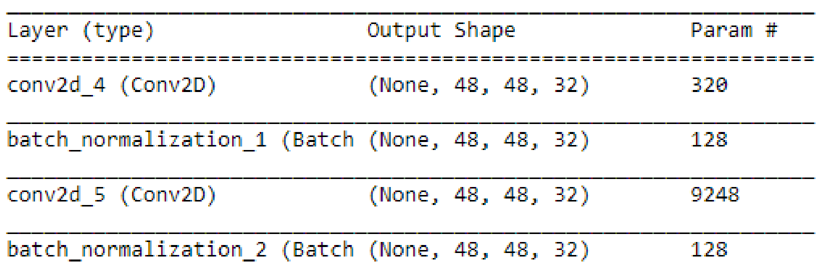
1. Preprocess and load the data (training, test and validation data) into numpy matrices as mentioned previously.
2. Create a model checkpoint to save the model weights at the end of each epoch only if the latest results are the best, otherwise the previous model will not be overwritten.
3. Create a convolutional neural network, the architecture of which is explained shortly, after description of these steps.
4. Next, we compile the model with inputs of loss, optimizer and the metrics that will be used to validate how well our model is training.
5. We finally start training the model with the following command:



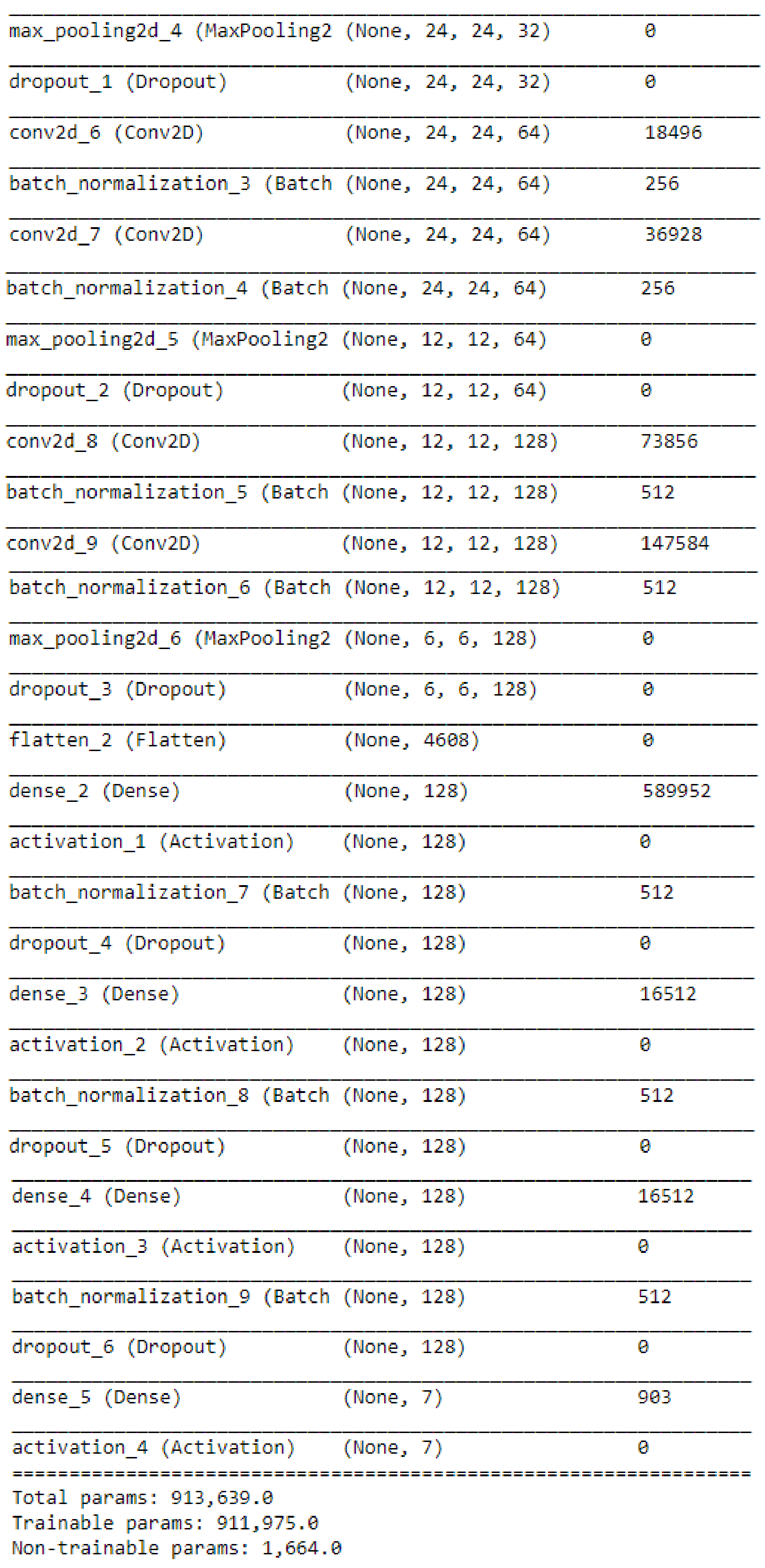
```
hist = model.fit(images_train, y_train_ohe, batch_size=128, epochs=5, verbose=1,
          validation_data=(images_val, y_val_ohe), shuffle=True, callbacks=[checkpoint])
```

6. Here we provide number of epochs and the batch size for training, with our training and validation data to calculate the accuracy at each step.
7. After this, we evaluate the model of test data and calculate the accuracy. If results are not satisfactory, we go to step 3 and change the parameters or layers and follow through till step 7.

The CNN architecture used is as follows:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_4 (Conv2D)            (None, 48, 48, 32)        320       
_________________________________________________________________
batch_normalization_1 (Batch (None, 48, 48, 32)        128       
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 48, 48, 32)        9248      
_________________________________________________________________
batch_normalization_2 (Batch (None, 48, 48, 32)        128       
```



```
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 24, 24, 32)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 24, 24, 32)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 24, 24, 64)         18496
_________________________________________________________________
batch_normalization_3 (Batch (None, 24, 24, 64)         256
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 24, 24, 64)         36928
_________________________________________________________________
batch_normalization_4 (Batch (None, 24, 24, 64)         256
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 12, 12, 64)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 12, 12, 64)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 12, 12, 128)        73856
_________________________________________________________________
batch_normalization_5 (Batch (None, 12, 12, 128)        512
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 12, 12, 128)        147584
_________________________________________________________________
batch_normalization_6 (Batch (None, 12, 12, 128)        512
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 6, 6, 128)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 128)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 4608)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)                589952
_________________________________________________________________
activation_1 (Activation)    (None, 128)                0
_________________________________________________________________
batch_normalization_7 (Batch (None, 128)                512
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)                0
_________________________________________________________________
dense_3 (Dense)              (None, 128)                16512
_________________________________________________________________
activation_2 (Activation)    (None, 128)                0
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)                512
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)                0
_________________________________________________________________
dense_4 (Dense)              (None, 128)                16512
_________________________________________________________________
activation_3 (Activation)    (None, 128)                0
_________________________________________________________________
batch_normalization_9 (Batch (None, 128)                512
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)                0
_________________________________________________________________
dense_5 (Dense)              (None, 7)                  903
_________________________________________________________________
activation_4 (Activation)    (None, 7)                  0
=================================================================
Total params: 913,639.0
Trainable params: 911,975.0
Non-trainable params: 1,664.0
```

In addition to previous explanation of layers, a basic understanding of other layers and parameters used is as follows:

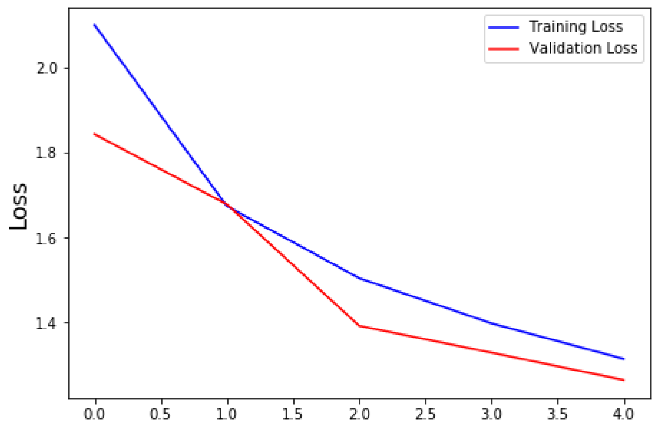
- **Flatten layer** converts the output of the convolutional part of the CNN into a 1D feature vector. It gets the output of the convolutional layers, flattens all its structure to create a single long feature vector to be used by let's say the dense layer for the final classification.
- **Batch normalization** is a technique of adjusting and scaling for improving the performance and stability of artificial neural networks.
- **Dropout** is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data.
- **Activation Layer** acts like an activation function that decides the final value of a neuron in a neural network. Suppose a cell value should be 1 ideally, however it has a value of 0.85, since you can never achieve a probability of 1 in CNN thus we apply an activation function.
    - The **ReLu** function returns 0 if it receives any negative input, but for any positive value xx it returns that value. So it can be written as f(x)=max(0,x).
    - The **softmax** function converts the outputs of each unit to be between 0 and 1 such that the total sum of the outputs is equal to 1.
- **Padding** is important because It's easier to design networks if you preserve height and width and don't have to worry too much about tensor dimensions when going from one layer to another

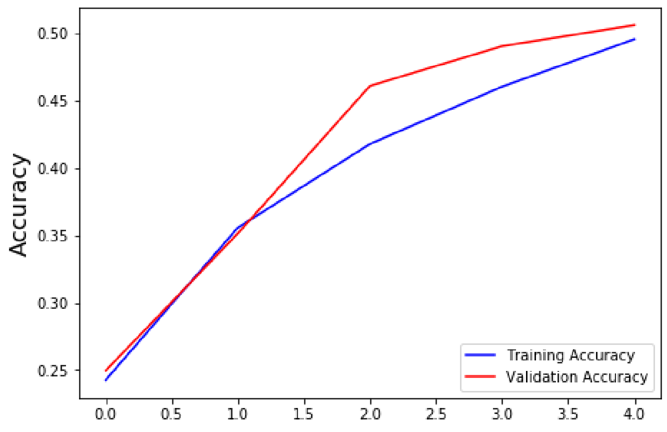
The first three layers that I have used have the same architecture:
Conv2D → Batch Normalization →Conv2D → Batch Normalization → MaxPool2D

The fully-connected layer at the end looks like this:
Dense → Activation → Batch Normalization → Dropout →
Dense → Activation → Batch Normalization → Dropout →
Dense → Activation → Batch Normalization → Dropout →
Dense → Activation

The test accuracy achieved here is 51.2%.

Visualization of accuracy and loss is:





Refinement

The motive of refining the above model here was to improve the test accuracy. I tried different layers and applied combinations of hyperparameters to the model to improve the accuracy and this section details what worked in favour of the purpose.

In the initial model, I have 2 convolutional layers of 32 units each, second two of 64 and last two of 128. Also, I used a dropout value of 0.3. I have not used any initializer in this model. I used Dense layer of 128 units. This gives me a test accuracy of 51.2%.

I tried changing various parameters here, but finally the one that showed positive results was applying a weight initializer to the layers. I chose random_uniform initializer which generates tensors with a uniform distribution. I also reduced Dropouts to 0.2 to ensure balance between overfitting and losing information processing on nodes.

After trying different parameters in layers, I increased the units to 64 in first two, 128 in second stage and 256 in the two convolutional layers of the third stage. After this, in all layers of the fully connected layer (except last Dense layer), I have used 256 units. I also trained the model longer for 9 epochs instead of just 5 that were done previously.

A significant change is that I used a weights initializer, random uniform.

The final refined model is explained in detail in the following section.

The final architecture gives 61.1% test accuracy which is an improvement on the previous model.

IV. RESULTS

Model Evaluation and Validation

The final model selected has the below architecture:

First Layer

1. Conv2D – 64 units, kernel initializer = random_uniform, activation – ReLu, padding - same
2. Batch Normalization
3. Conv2D - 64 units, kernel initializer = random_uniform, activation – ReLu, padding - same
4. Batch Normalization
5. MaxPool2D – pool size – (2, 2), strides - 2
6. Dropout – 0.2

Second Layer

1. Conv2D – 128 units, kernel initializer = random_uniform, activation – ReLu, padding - same
2. Batch Normalization
3. Conv2D – 128 units, kernel initializer = random_uniform, activation – ReLu, padding - same
4. Batch Normalization
5. MaxPool2D – pool size – (2, 2), strides - 2
6. Dropout – 0.2

Third Layer

1. Conv2D – 256 units, kernel initializer = random_uniform, activation – ReLu, padding - same
2. Batch Normalization
3. Conv2D – 256 units, kernel initializer = random_uniform, activation – ReLu, padding - same

4. Batch Normalization
5. MaxPool2D – pool size – (2, 2), strides - 2
6. Dropout – 0.2

Full Connected Layer 1

1. Flatten
2. Dense - 256 units
3. Activation – ReLu
4. Batch Normalization
5. Dropout – 0.2

Fully Connected Layer 2

1. Dense - 256 units
2. Activation – ReLu
3. Batch Normalization
4. Dropout – 0.2

Fully Connected Layer 3

1. Dense - 256 units
2. Activation – ReLu
3. Batch Normalization
4. Dropout – 0.2

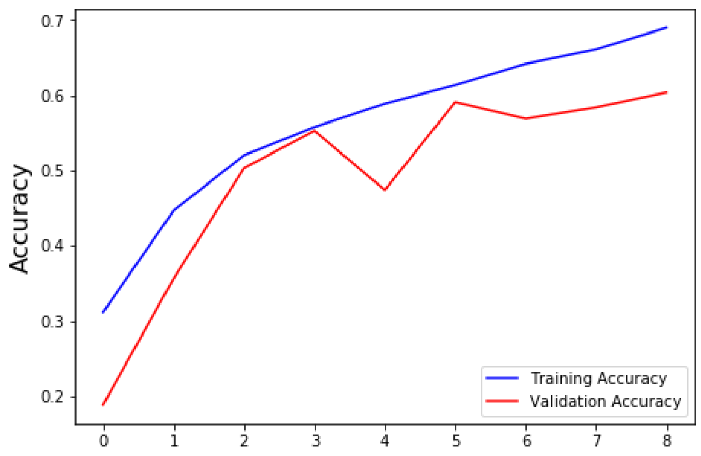
Final Layer

1. Dense – 7 units
2. Activation - softmax

The results achieved are:

```
Training Accuracy:  0.689923020689
Validation Accuracy:  0.603789356371
3589/3589 [==============================] - 255s
Test score: 1.09054092774
Test accuracy: 0.611033714143
```

As shown, **test accuracy** achieved is **61.1%**.



Validation accuracy as seen above does falter a bit while learning but resumes track of improvement.

After this, I did a final testing on external data for visual confirmation of the results, to check the robustness of the model.

The test showed that 3 out of 4 images were correctly predicted for facial expression. This is further elaborated on in subsequent sections.

Justification

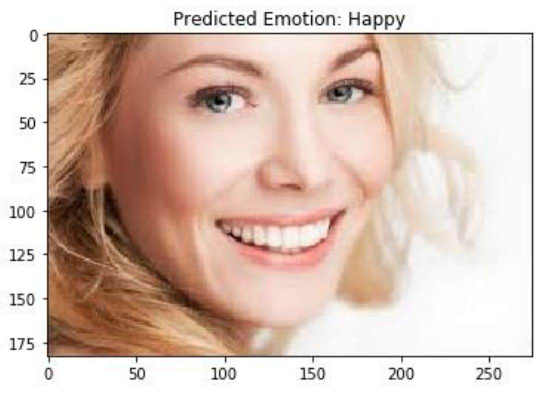
As compared to the benchmark model of a simple CNN created that produced 41% accuracy, this model has shown vast improvement by resulting in a test accuracy of 61.1%. Even though this is lower than the results produced by Charlie Tang in the Kaggle competition, with 71% accuracy, the model has performed well considering limitations presented pertaining to processing power by my laptop, time constraints, and the scope of the project. The results are satisfactory as the model was also tested against images external to the data set.

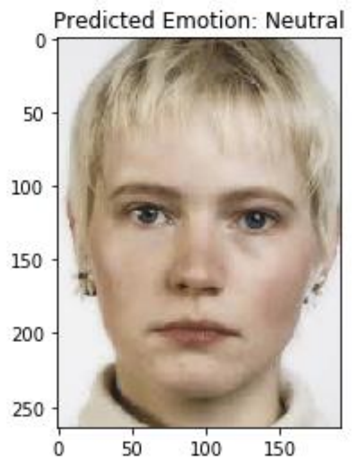
The model performed better than average on the input images and could correctly classify 3 out of 4 images, external to the dataset. It does get confused between the specifics of the emotions, e.g. surprised face was predicted as happy, as the facial points are quite similar for those two emotions. The accuracy achieved is acceptable for the purpose of this problem and will do well on basic emotion prediction projects. Moreover, this is a stepping stone for advanced image classification and emotion recognition models.

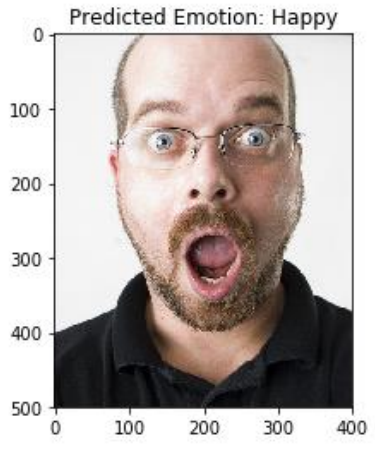
V. CONCLUSION

Free Form Visualization

The following are the images that were processed and correctly/incorrectly classified.







The fourth image's expression is misclassified as 'Happy' whereas the correct emotion is 'Surprised'.

The model is not able to differentiate well between the nuances of facial expressions that are extremely close to each other. FER is quite a difficult problem to solve as every person expresses differently and it is quite a difficult task for even a human to read a person's expressions to understand the emotion.

Reflections

The steps I have used to implement this project in a nutshell are:

1. Download the data into folders organized by training data, validation data and test data.
2. Preprocess the images to ensure that the size is 48x48 pixels and grayscale is used for easier processing. Image pixel arrays are divided by 255 and dimensions are expanded so that data is appropriate to be fed to the model.
3. A simple CNN is created to compare the accuracy of the final model with.
4. A model is implemented by trying different layers and parameters that is much more sophisticated than the one used in step 3. This model presents a much better accuracy.
5. The model in step 4 is refined further by adjusting different parameters and results in the final refined model that is used as a solution to the problem. This solution results in 61% accuracy.
6. This model is tested on external images to confirm visually, the robustness of the implementation.

Steps 4 and 5 presented with maximum difficulty as it took me hours to train the models, given the hardware I am using is just a basic laptop. However, considering these constraints, the model has performed well to my satisfaction as it was able to predict the expression on external data set quite well.

I learned in depth lot of computer vision techniques, including OpenCV and also learned about weight initializers, and saw the differences with the help of implementation. I saw that a few times my model started to overfit, and I used Batch Normalization and Dropout to control the same.

The maximum learning from the project is that I got familiar with how to present a problem, find datasets, write a solution toward solving the problem, which at the beginning of the nanodegree, I was very nervous about.

A very interesting point is that I had initially forgotten about dividing the pixels by 255, so as to normalize the changes caused to the gradient signal while back propagation. This boosted my accuracy from 57% to 61%.

Improvement

For further improving the project, the following can be tried:

1. Longer training on a GPU will help much better in getting more accurate results. Perhaps a cloud instance will better serve the purpose.
2. Using regularizers and bias initializers will further help improve the accuracy.
3. A different approach could be trying transfer learning, as that will help improve the accuracy.

VI. REFERENCES

1. A Brief Review of Facial Emotion Recognition Based on Visual Information
2. Challenges in Representation Learning: Facial Expression Recognition Challenge
3. Deep Learning using Linear Support Vector Machines
4. Static facial expression recognition with convolution neural networks
5. Udacity MLND Report Example 1
6. Image classification with CNN