Machine Learning Engineer Nanodegree

Capstone Project

Keshri Nandan
August 27th, 2017

Project Overview

Anyone who has watched the American TV show “Lie to me”, would share my fascination of reading people’s emotional state from their faces alone. While most people are naturally able at this, machines haven’t yet advanced enough to match a human at this skill. As AI makes great advances in performing repeatable tasks with perfection, strategizing in complex games etc., emotional intelligence still remains an unexplored domain.

As we build smarter systems, contextual awareness becomes more and more important. The emotional state of the user becomes an important factor in defining the context. Reading emotions directly of the faces can be huge leap and can be beneficial in multiple applications:

1. Intelligent homes: house lighting, music etc. can be customized to the mood
2. Personalized marketing: The ad campaigns can be better targeted with the right emotional state in mind
3. Surveillance: surveillance systems could become smarter in predicting events as in “Person of Interest”

Emotions (the seven basic ones) surface on faces in universally characteristic manner. These characteristic features can be leveraged to recognize emotions on any face.

Anger Disgust Fear Happy Sad Surprise Neutral

This topic has garnered great amount of interest in the recent past. It became all too popular when contests were hosted on Kaggle and EmotiW2015 to devise efficient algorithms to identify facial emotions. Multiple approaches like CNN, SVM, SIFT & MKL have shown great results. CNNs have proved to the most promising in terms of improving the accuracy further.

Problem Statement

The most common emotions have similar facial patterns. Given the image of a face, the objective is to identify the emotion expressed by the face. The task is to categorize the faces based on the emotion shown in the facial expression in to one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral)

Solution Approach

With characteristic features evident from faces across the globe, it suits well to extract them to identify the emotions.

The pixels in close proximity are more related with each other than with pixels that are a greater distance away. This **spatial structure** of the image needs to be leveraged.

While these patterns are universal, they surface on the faces at different locations, thus **shift invariance** is required of the solution we adopt. From the discussion it is evident that we are solving a **classification** problem here.

All of these conditions lead us to use **Convolutional neural networks**. We shall be starting with a base structure and build on it to improve its performance by tuning its hyper-parameters.

Analysis

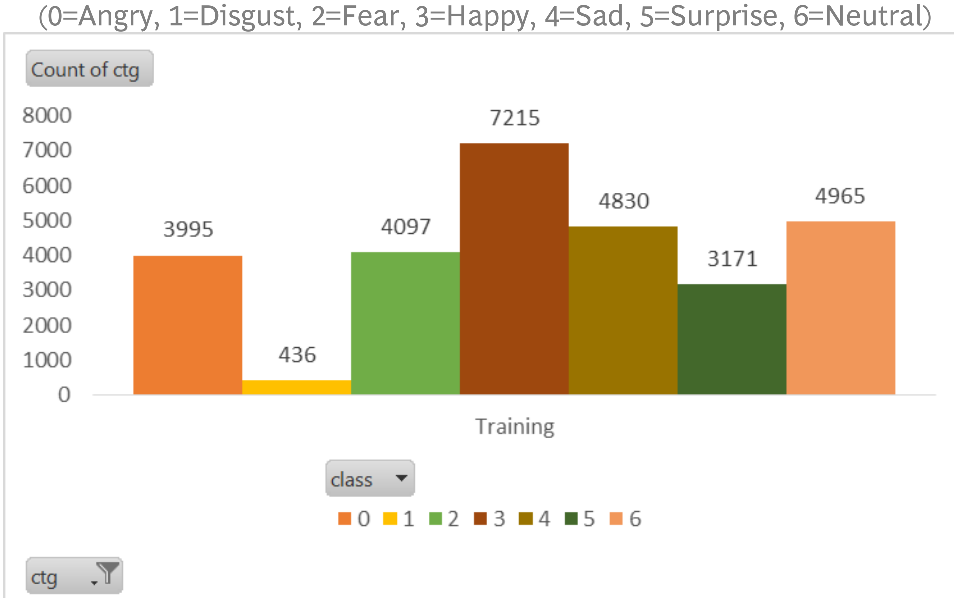
Datasets and Inputs

The data to be used is the **Fer2013** dataset shared in the Kaggle contest: “Challenges in Representation Learning: Facial Expression Recognition Challenge” It has around 35000 facial images in 48x48 greyscale pixel format and the associated emotion as label. There are 7 standard emotions identified in the dataset. The dataset contains two columns, “emotion” and “pixels”. The “emotion” column contains a numeric code ranging from 0 to 6, inclusive, for the emotion that is present in the image. The “pixels” column contains a string surrounded in quotes for each image. The contents of this string a space-separated pixel values in row major order.

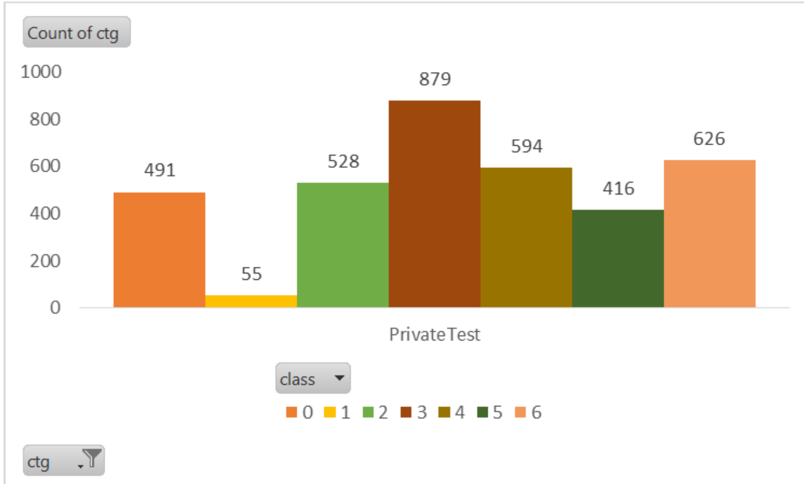
The images are divided under 3 categories:

1. Training: It has 28,709 images spread across 7 emotions as below:

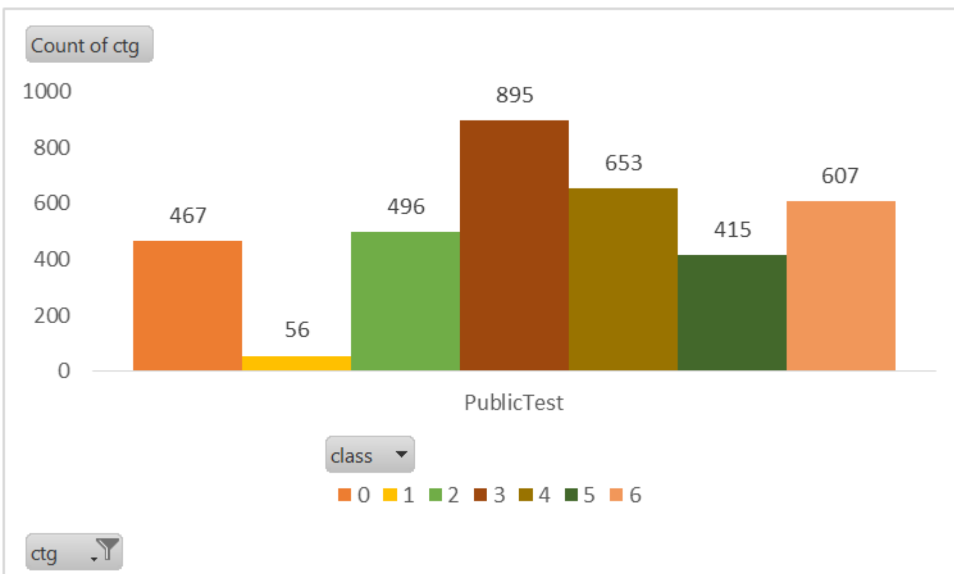
(0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral)



2. Validation: It has 3589 images distributed as:



3. Test: It has 3589 images distributed as:

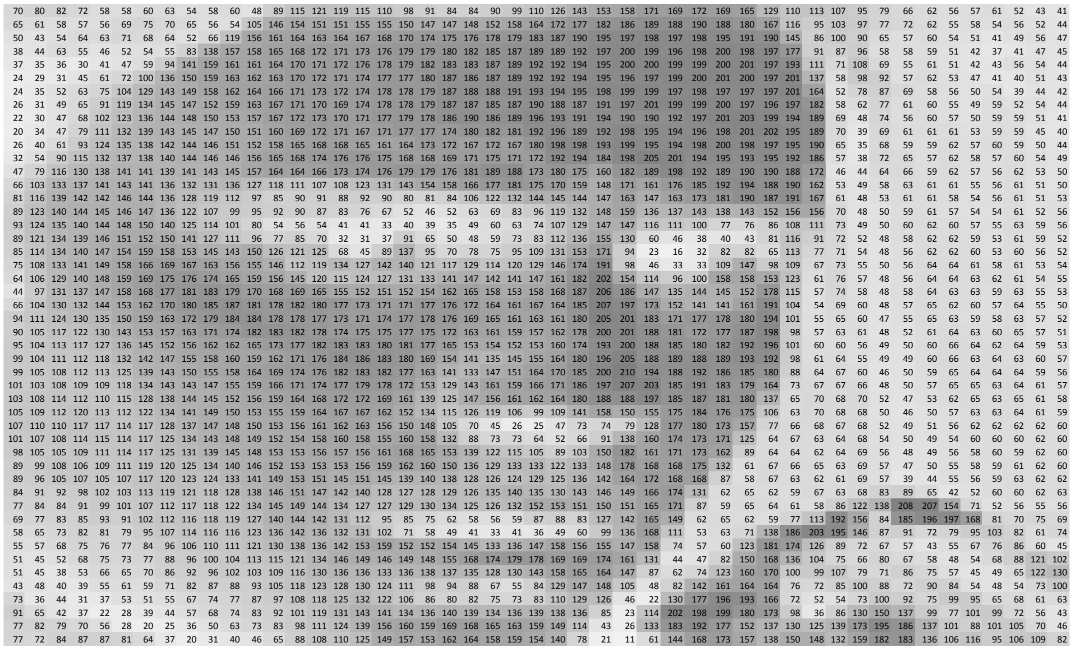


Data pre-processing

The dataset is hosted on the Kaggle competition's ("Challenges in Representation Learning: Facial Expression Recognition Challenge") page. It has three columns:

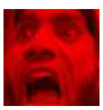
1. "Emotion" – indicating emotion of the face
2. "pixels" – it is a 2304X1 array of pixels of the 48X48 greyscale image
3. "category" – it segregates the dataset into:
    a. Training
    b. PrivateTest: used for validation
    c. PublicTest: used as test set

The 48X48 grid for a face looks like this:

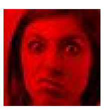


The pixels are used to generate the images (using PIL) and arrange them in 7 folders for the 7 emotions across three folders (Training, PrivateTest, PublicTest)

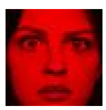
Anger



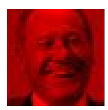
Disgust



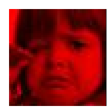
Fear



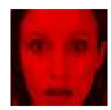
Happy



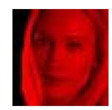
Sad



Surprise



Neutral



The Kaggle dataset used here includes pixels for grayscale only, so only one channel is available.
The images used here have the faces only, so no localization or cropping is required.

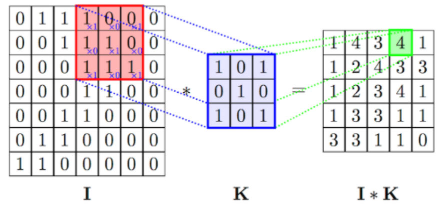
The dataset provides the pixel values in 2304X1 array, which are reformatted into 48X48 grid and the images are then generated. These images are automatically arranged in the folder structure provided by the first block of code.

Methodology

It is a classification problem and involves extracting features from images as inputs.
Convolutional neural networks (CNNs) have proven to be robust in such situations. Convolutional Networks are similar to Neural Networks where neurons are replaced by the convolutional operation at the initial layers. A typical CNN has three type of layers:

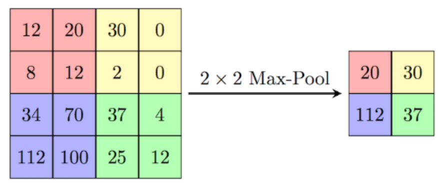
1. <u>Convolution layers:</u>

   Convolutional layers consist of a rectangular grid of neurons. It requires that the previous layer also be a rectangular grid of neurons. Each neuron takes inputs from a rectangular section of the previous layer; the weights for this rectangular section are the same for each neuron in the convolutional layer. Thus, the convolutional layer an image convolution of the previous layer, where the weights specify the convolution filter.

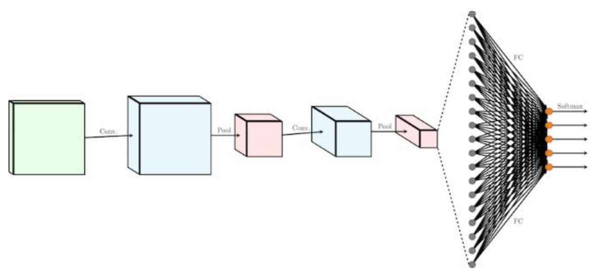


Previous layer Neuron Convolution layer

2. <u>Pooling layers:</u>
   After each convolutional layer, there may be a pooling layer. The pooling layer takes small rectangular blocks from the convolutional layer and subsamples it to produce a single output from that block. There are several ways to do this pooling, such as taking the average or the maximum, etc. Our pooling layers will always be max-pooling layers; that is, they take the maximum of the block they are pooling.
   A diagrammatical illustration of 2×22×2 max-pooling is given below

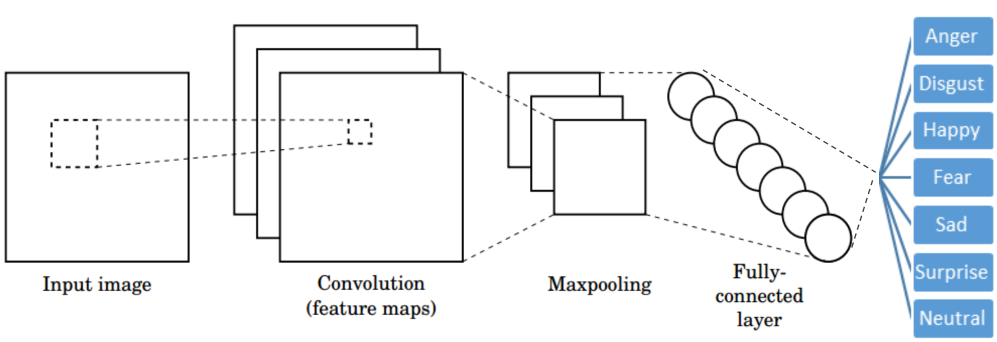


3. <u>Fully connected layers</u>:
   After several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. A fully connected layer takes all neurons in the previous layer (be it fully connected, pooling, or convolutional) and connects it to every single neuron it has. Fully connected layers are not spatially located anymore.



Input – Conv – MP – Conv – MP – Fully connected layer

The architecture of a CNN is designed to take advantage of the 2D structure of an input image. This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units.



Evaluation Metrics

1. Accuracy: The accuracy of predicting which of the seven classes should be assigned to each image

   The classification accuracy Ai of an individual program i depends on the number of samples correctly classified (*true positives plus true negatives*) and is evaluated by the formula:

$$\text{Ai} = \left(\frac{t}{n}\right) * 100$$

   Where t = number of sample cases correctly classified
   n = total number of sample cases

   Accuracy is the most readily comprehensible metric even to a layman. The dataset has fair representation for all classes except the (1 = Disgust) class. Hence we can use this as the performance metric.

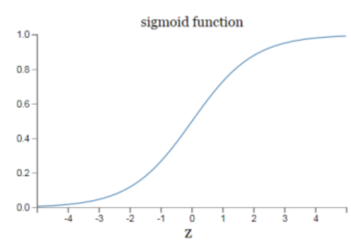
2. Optimizer – 'rmsprop': The learning rates for the weights are configured to be tuned by dividing the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight

   While training the CNN, with increasing epochs the learning rate needs to be scaled with the gradients as too small steps shall slow down the training progress.

3. Loss function – 'categorical crossentropy': For classification problems the standard way of calculating losses (to be minimized) is by evaluating the cross-entropy between the predicted classes and true classes
   Mathematically, it is defined as:

$$H(p, q) = -\sum_{x} p(x) \log q(x).$$

   Where, p(x) are the true classes and q(x) probabilities of predicted classes



   For neural nets, the learning rates are dependent on (∂C/∂w) and (∂C/∂b) where:
   C = cost function, w = weights and b = biases.
   If the cost function is the cross-entropy function, these terms are directly proportional to the error (σ(z)-y) i.e. (predicted – actual). This ensures that learning at initial stages is not exceedingly slow as incase of the quadratic cost function where these terms are proportional to σ′(z) (sigmoid)

Benchmark Models

1. Yu & Zhang reported accuracy of 61% in the EmotiW2015 challenge. They used CNN with **randomized perturbation**
2. Mollahosseini and Mahoor used **inception layers with convolution layers** and reported varying accuracy (60% - 94%) across multiple datasets (FER2013 – 66%)
3. Kim, Roh, Dong & Lee trained multiple deep **convolutional neural networks** (**deep CNNs**) **as committee members and combined their decisions** for solving a seven-class problem of static FER in the wild for the EmotiW2015, and achieved a test accuracy of 61.6 %.

Implementation

The code structure is explained below:

The python notebook “emoticon” is structured in the steps we take in the process of developing the model.

1. Step 0: Prepare image repository
   This section of the code reads in the dataset (fer2013.csv) row-wise and generates the images and saves them in the associated folder structure.
   This needs to be run only once while setting up the project.
2. Step 1: Import image data
   The image paths are then read into model according to their category (Train, test, validate)
   Their associated emotion classes are read as categorical variables across the three categories
   The next cell defines the functions that read in the image from the image path and return their corresponding 4D tensors
3. Step 2: Rescale image data
   The pixel data is rescaled from 0-255 to 0-1 by dividing it with 255
4. Step 3: Define the model architecture
   The initial model and further tuning to it are employed here. After defining the model it is compiled and trained in the subsequent cells. While training, the model is configured to save the learning history and the best set of weights.
   After we have trained the model, the best set of models are loaded.
   For every tuning iteration, steps 3 and 4 are repeated
5. Step 4: Test the model
   The model id now tested with the “test” dataset and its accuracy is recorded.
6. Step 5: Model prediction
   The final step involves generating sample predictions from the model and comparing with actual classes

   Please note:

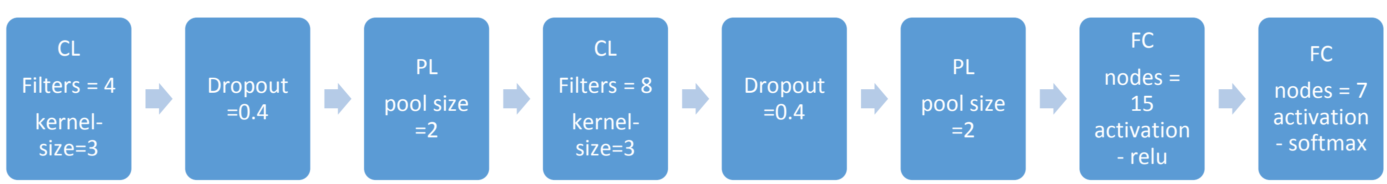
- While training the model in step 3, it is advisable to have the “verbose” option set to 1 if one is running it for 10-15 epochs as it helps keeping track of the training and validation errors.
  Beyond 20 epochs, one must set it to 0 as it slows down the system (it did for me). To keep track of errors and accuracies, one should record these in a log using the “**CSVlogger**” callback.
- The model also uses the “**ModelCheckpoint**” callback. This option saves the weights of the model for every epoch. It has been configured to save the best set of weights i.e. only when the validation loss improves.

  Keras offers multiple callbacks one could leverage to diagnose one’s model.

Refinement

1. **Model 1:** We begin with a simple CNN with 2 convolution layers (CL), 2 pooling layers (PL), 2 fully connected layers (FCL) : Epochs = 50

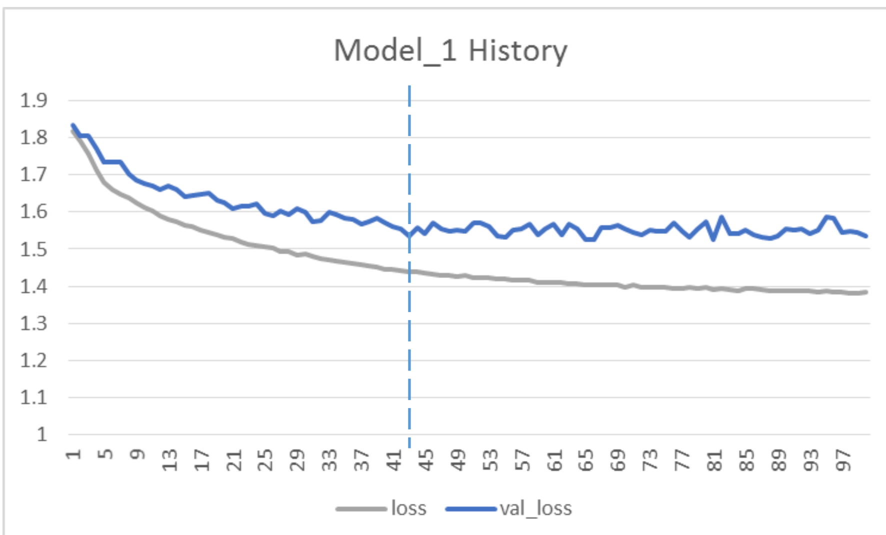
   Weights (for CL) initialized with std-dev. = 0.01



```
________________________________________________________________
Layer (type)                 Output Shape              Param #
================================================================
conv2d_1 (Conv2D)            (None, 48, 48, 4)         112
________________________________________________________________
dropout_1 (Dropout)          (None, 48, 48, 4)         0
________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 4)         0
________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 8)         296
________________________________________________________________
dropout_2 (Dropout)          (None, 24, 24, 8)         0
________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 8)         0
________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
________________________________________________________________
dense_1 (Dense)              (None, 15)                17295
________________________________________________________________
dense_2 (Dense)              (None, 7)                 112
================================================================
```

If we follow the loss (training vs validation) history over training epochs, we observe that the validation error flattens out at around epoch = 55.

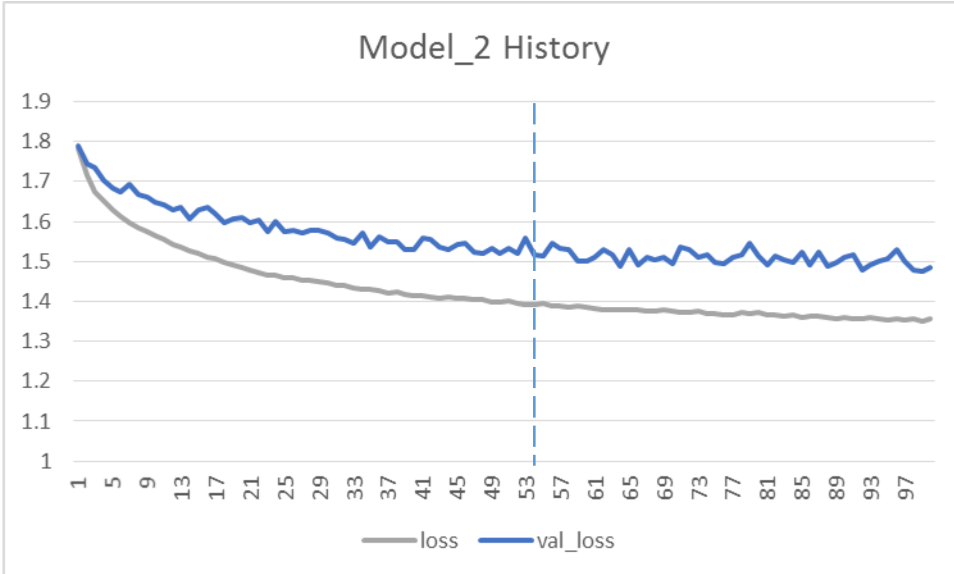
The best set of weights are saved in the folder 'saved_models'. We load these weights to check the accuracy of the model.



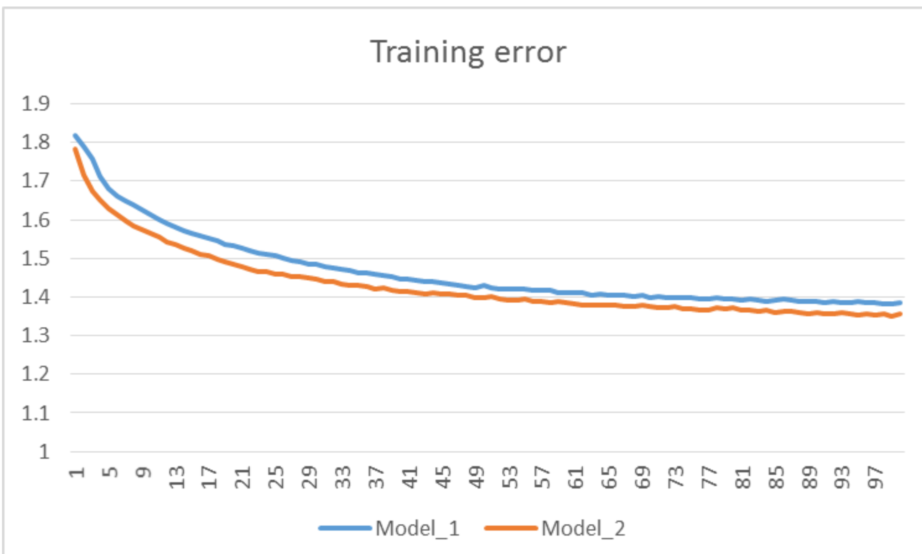
Test accuracy - 38.9245%

This lends us a good starting point to begin with. We shall tweak other parameters to further improve the accuracy of the model

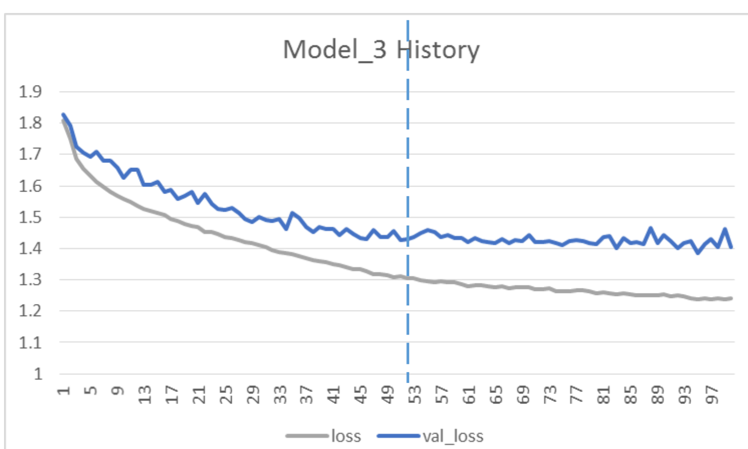
2. **Model 2:** In the next iteration we use the same architecture but initialize the CL weights by Xavier initialization (glorot_normal)
In this case the validation loss flattens out around epoch = 65.
The test accuracy jumps to **43.1039%**



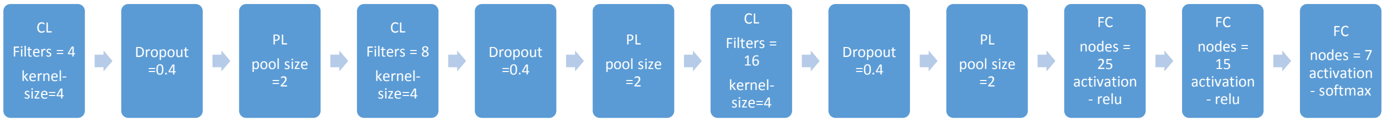
If we compare the training error for model_1 & model_2, we see improvement that reflects in the test accuracy.



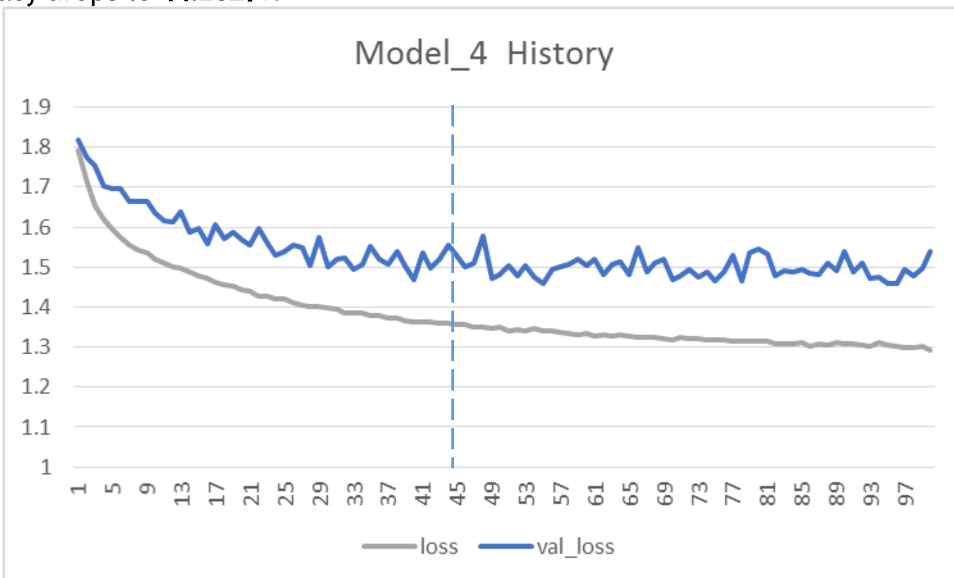
3. **Model 3:** Adding another fully connected layer (FC-25) before (FC-15), the model improves and test accuracy rises to **46.8376%**



4. **Model 4:** We continue with model 3 with modifications
    a. Add a convolution layer + pooling layer
    b. Kernel sizes: 4 – 4 – 4

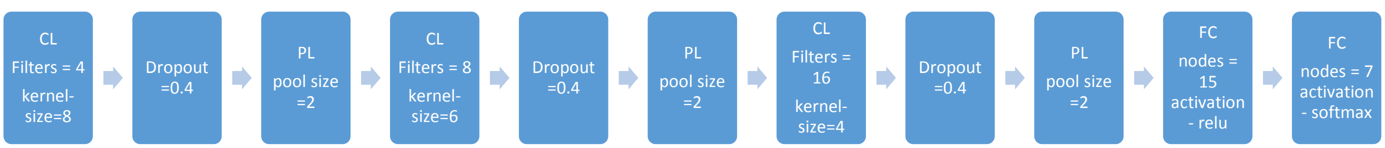


The test accuracy drops to **44.1627%**

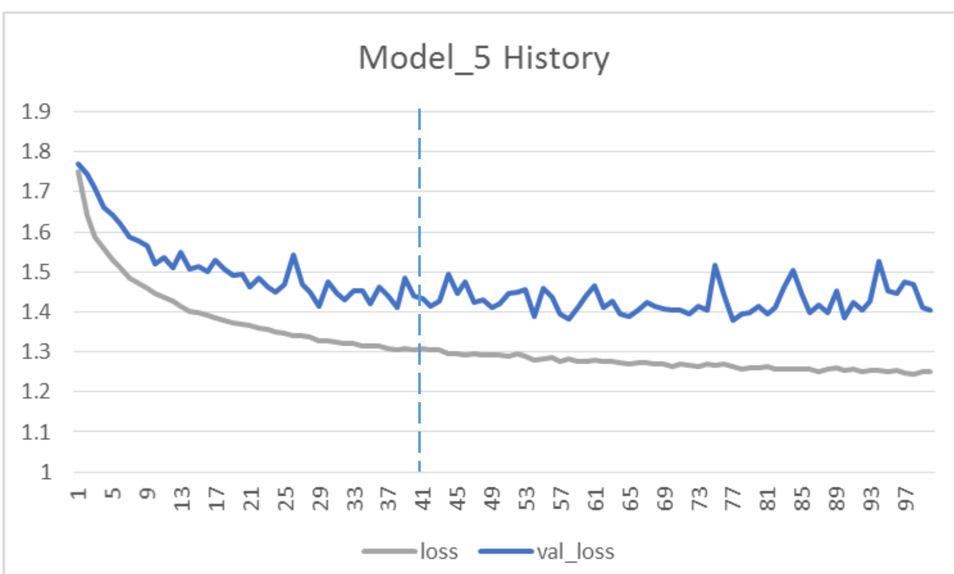


5. **Model 5:** We continue with model 4 with modifications

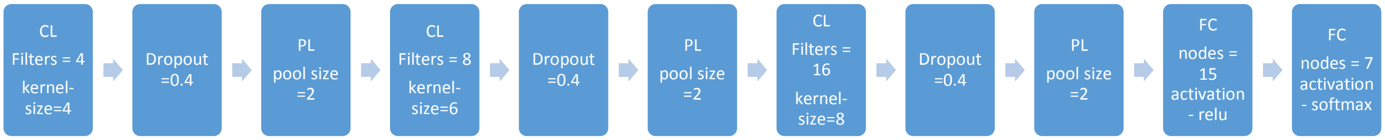
    a. Modify kernel-size progressively (8 – 6 – 4)



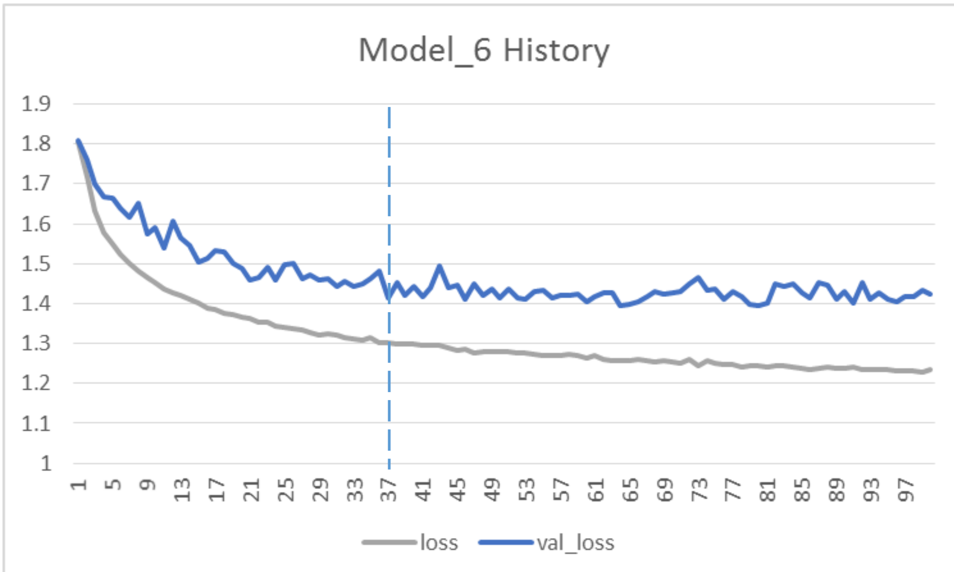
The test accuracy is reported to be **46.9211%**



6. Model 6: We continue with model 4 but reverse the kernel sizing scheme (4 – 6 – 8)



This pushes the test accuracy to **47.5899%**



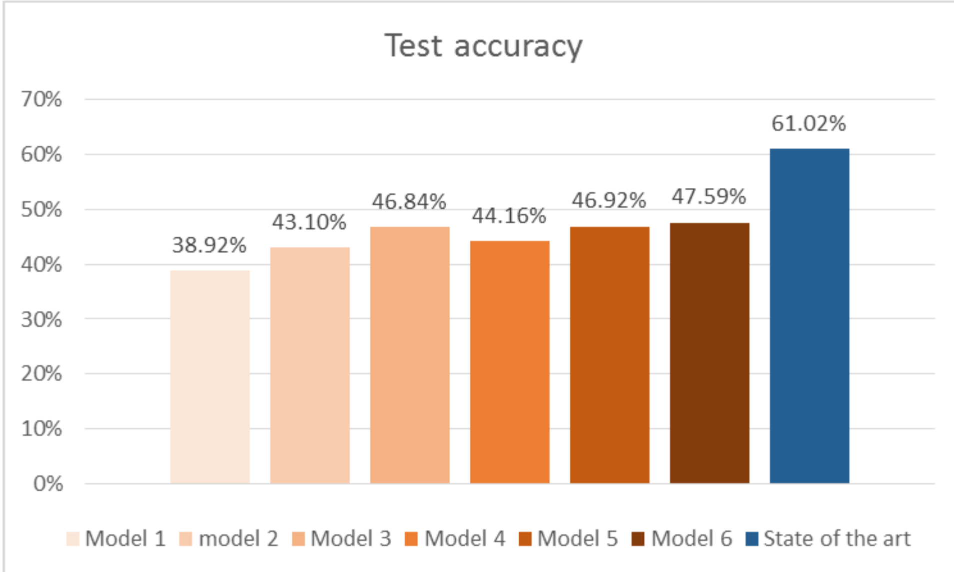
Results

From the training history it also comes out that, as the model performance improves, the validation loss plateaus at an earlier epoch:

Model	Epoch	Accuracy
Model – 4	45	44.1627%
Model – 5	41	46.9211%
Model – 6	37	47.5899%

Model – 6 has the maximum accuracy achieved in this exercise which falls short of the state of the art accuracy (**61%**) by **13 pts.**

The best models are fairly deep as compared to the Model_6 and require greater computational resources than a laptop.
The objective of this exercise was to test out novel architecture schemes and gain significant performance gains with these successive tweaks. Further experimentation may result in better schemes that can improve existing models.



Model-6 description

The model employs varying size of kernels (4 – 6 – 8). This might help us extract multiple levels of abstraction from the images to better capture features specific to emotions. Further tuning of the kernel-sizes and their counts can help us achieve the optimum configuration.

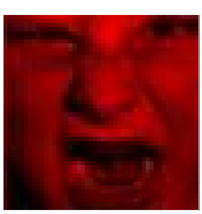
For this model the best weights are saved for epoch = 54, where validation accuracy = 45.0822%, and train accuracy = 48.9359%. Since the accuracies are within the same range (45% to 49%), and training accuracy slightly greater than validation accuracy, we can rule out overfitting.

The current value of dropout (0.4) is doing a good job of regularizing the model as we see multiple troughs and ebbs in the validation error curve unlike the monotonous decline of training error. We can explore model's performance with multiple values of dropouts.

The learning curves for each iteration have been depicted with the epochs highlighted where validation errors flatten out. We have configured the model to save the best weights, this ignores the set of weights that may result due to overfitting.

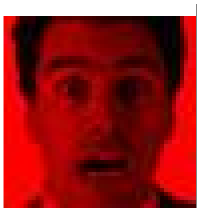
Conclusion

Free-form visualization

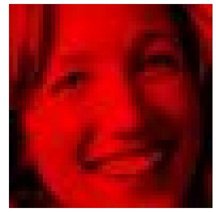


```
Emotion detected: Happy
Actual emotion: Angry
```

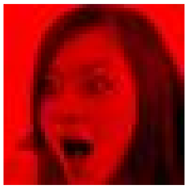
```
Emotion detected: Neutral
Actual emotion: Disgust
```



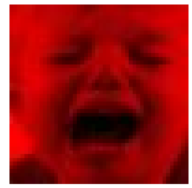
```
Emotion detected: Fear
Actual emotion: Fear
```



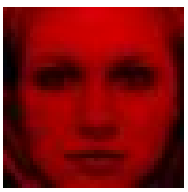
```
Emotion detected: Happy
Actual emotion: Happy
```



```
Emotion detected: Surprise
Actual emotion: Surprise
```



```
Emotion detected: Angry
Actual emotion: Sad
```



```
Emotion detected: Neutral
Actual emotion: Neutral
```

From above results we can summarize that,

- Characterizing emotions on children's faces can be tricky. ☺
- "Disgust" if expressed subtly can hard to distinguish from "Neutral" faces
- The model fares fairly well for adult faces irrespective of gender

Reflections

The steps taken in the project can be summarized in the following points:

1. An interesting problem with readily available dataset was found.
2. The dataset was used by multiple teams in past, so the benchmark was well established.
3. The methodology was settled on to with evident reasoning
4. A basic model was further refined by applying best practices and tuning the parameters
5. The best model achieved was then chosen after it consistently resulted in high accuracies

The difficult part of the process was to manually tune the architecture of the model and run multiple iterations of these versions. While parameter optimization techniques exist, they proved to be computationally expensive and don't warrant novel schemes.

The interesting part of the project was that something as trivial/natural for humans was not evident for machines and that it is part of active research across the AI community. Seeing something out of sci-fi becoming real was fascinating.

The exercise has been a great learning experience and has sparked my curiosity to employ ML techniques in various aspects of my work.

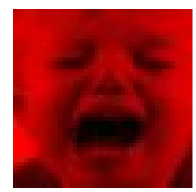
Scope of improvement

There are multiple avenues to explore to improve the accuracy of the model.

1. Augmentation of the training dataset
   a. The human face being symmetric across the vertical axis, adding laterally inverted images in the training set might improve accuracy
2. Hyper-parameter optimization
   a. Currently the manual approach has been followed as I have limited computational resources (no GPUs)
   b. One could employ grid-search cross-validation, Gaussian process etc.
   c. These are resource intensive processes and, hence manual tuning was employed
   d. It is recommended to use SKlearn with Keras to deploy **GridSearchCV**
3. Explore deploying the deep forest model
   a. The model has shown promising results in deep learning problems
   b. This could not be attempted owing to paucity of time
4. Fractional max-pooling and batch-normalization
   a. These concepts have been tested by Raghuvanshi and Choksi [10] and achieved similar accuracies
   b. These can be tested with the current architecture to see if performance improves

While we could improve our model in various ways, we need to reassess our initial assumption that all emotions surface in the same fashion universally.

The definition of emotion appears to be malleable with age-groups. For example, the image of the kid classified as "Sad" could easily be construed as "Angry" for an adult.



Emotion detected: Angry
Actual emotion: Sad

References

1 https://www.humintell.com/2010/06/the-seven-basic-emotions-do-you-know-them/
2 http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/
3 Image based Static Facial Expression Recognition with Multiple Deep Network Learning http://www.contrib.andrew.cmu.edu/~yzhiding/publications/ICMI15.pdf
4 Going Deeper in Facial Expression Recognition using Deep Neural Networks https://arxiv.org/pdf/1511.04110.pdf
5 Hierarchical committee of deep convolutional neural networks for robust facial expression recognition https://link.springer.com/article/10.1007/s12193-015-0209-0
6 Challenges in Representation Learning: Facial Expression Recognition Challenge https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge
7 Deep Forest: Towards An Alternative to Deep Neural Networks https://arxiv.org/abs/1702.08835
8 http://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/
9 https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html
10 http://cs231n.stanford.edu/reports/2016/pdfs/023_Report.pdf