

Rudraksh Tuwani

1 June 2017

# Emotion Recognition Using Deep Convolutional Networks



# I. Definition

## Project Overview

The project is in the domain of computer vision and emotion recognition. Broadly speaking, computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images whereas emotion recognition is about identifying the emotion expressed in images of human faces, voice recordings etc.

The goal of this project is to develop a model to detect and classify the emotion expressed in static images of human faces into seven broad categories using computer vision techniques. Researchers have increasingly been shifting from rule based approaches to machine learning based approaches for a variety of computer vision tasks. Convolutional Neural Networks (CNNs), a type of neural networks, have recently achieved the state-of-the-art in a variety of sub domains of computer vision and were thus chosen for this particular task.

I used the fer 2013<sup>[1]</sup> dataset for training the CNN, which was made available on Kaggle as part of a competition. The competition was conducted back in 2013, with more than 56 teams comprising of leading researchers and graduate students of computer vision taking part. More details regarding the leaderboard are available in the *benchmark* section.

## Problem Statement

The data consists of 48x48 pixel grayscale images of faces. The task is to categorise each face based on the emotion shown in the facial expression into one of seven categories, namely:

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

Besides centring of faces, the dataset hasn't been preprocessed much and contains images with variable lighting, different poses, and even watermarks in some cases.

Thus, the combining of low quality images with a lot of noise of the above form, makes this a pretty challenging problem even for humans.

A brief of the solution is as follows:

1. Data Cleaning - This is the first step and involves detecting as well as removing corrupted images from the dataset.
2. Data Augmentation - The size of the dataset is on the lower side when it comes to using it to train neural networks. I used data augmentation techniques to artificially increase the size of the dataset. This also made the model more robust.
3. Building a classifier - As mentioned previously, I trained a Convolutional Neural Network for the task of classifying emotion.

## Metrics

I used multi class cross entropy as the evaluation metric. It is defined as follows:

$$H_{y'}(y) = - \sum_i y'_i \log(y_i)$$

where  $y$  is our predicted probability distribution, and  $y'$  is the true distribution (the one-hot vector with the digit labels).

Multi class cross entropy is pretty much the industry standard for training Neural Networks for classification tasks. Intuitively, the cross-entropy is measuring how inefficient our predictions are in describing the truth. It takes a high error value when the model makes big mistakes (like giving high probability for the wrong emotion etc.), and a relatively lower value when the model isn't wrong by much (like giving the second highest probability for the correct emotion etc).

Additionally, I'll also be providing accuracy as a more raw and intuitive measure. Accuracy is simply defined as the ratio of number of cases we get right to the total number of cases tested.

## II. Analysis

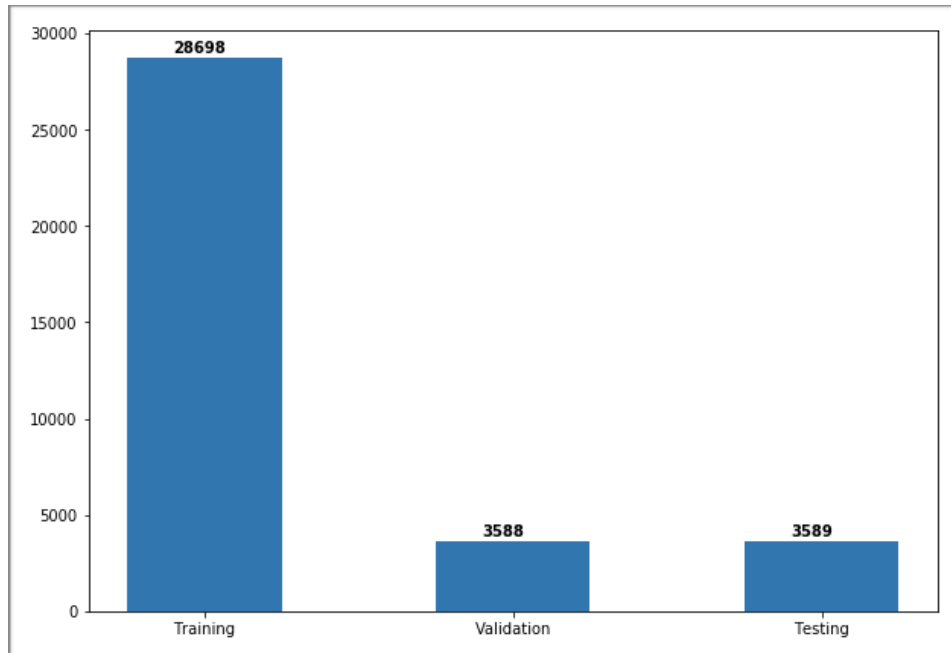
### Data Exploration

The fer 2013 dataset contains 48 x 48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. The data is available in the form of a CSV file with three columns:

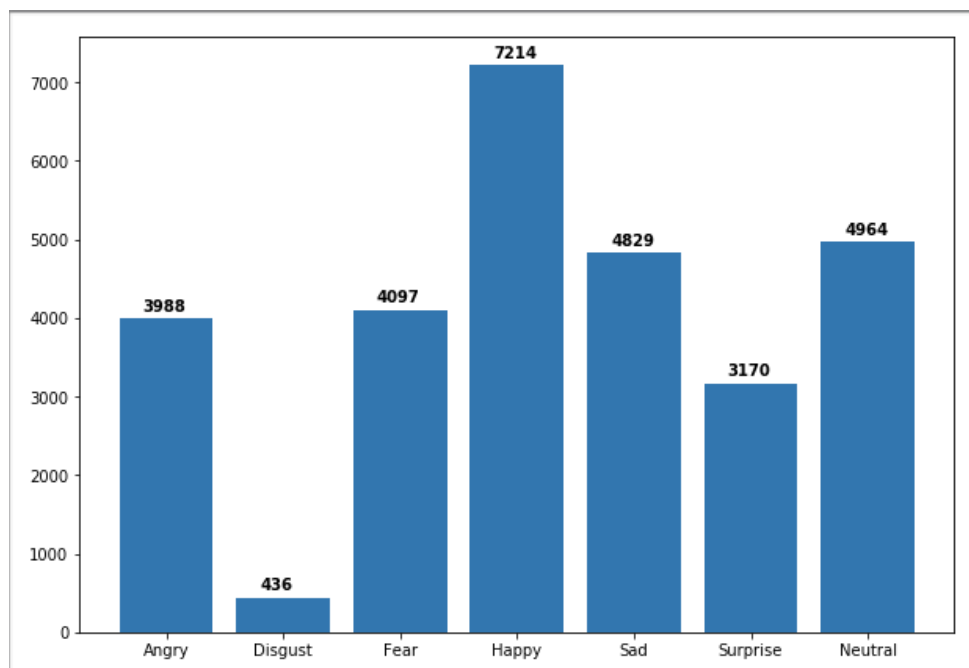
1. emotion - Contains a numeric code ranging from 0 to 6, inclusive, for the emotion that is present in the image. number from 0 to 6 specifying the emotion expressed in the image.
2. pixels - Contains a string surrounded in quotes for each image.
3. usage - Specifies whether the image was used for "Training", "PublicTest" or "PrivateTest" on Kaggle.

The total number of the images in the dataset is 35,875. Some of the images were found to be corrupted and were removed in the “Data Preprocessing” step. The following split was performed on the cleaned dataset:

- Training set: 28,698 images labelled as training in usage column.
- Validation set: 3,588 images labelled as PublicTest in usage column.
- Testing set: 3,589 images labelled as PrivateTest in usage column.



We have a total of seven (7) classes in our dataset. This distribution of training images over the classes are as follows:



As we can see from the above plot, the classes are not very well balanced. The emotion “Disgust” has just 436 images in our training set. Images for different emotions are previewed below:

### Angry



### Disgust



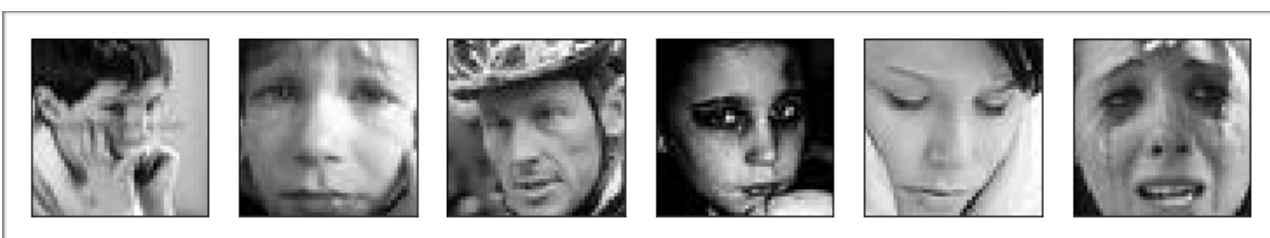
### Fear



### Happiness



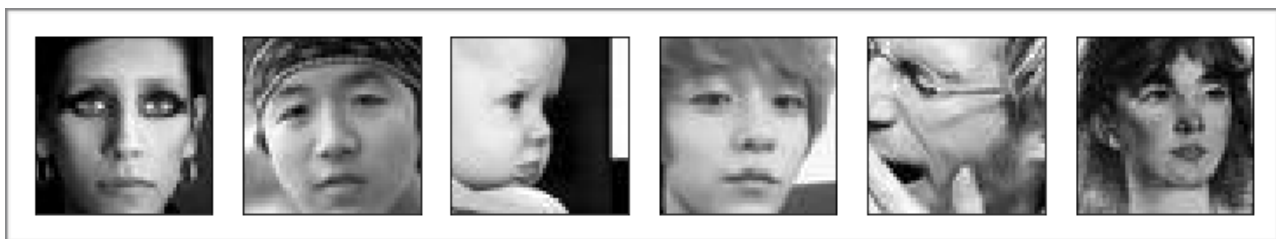
### Sad



### Surprise



### Neutral

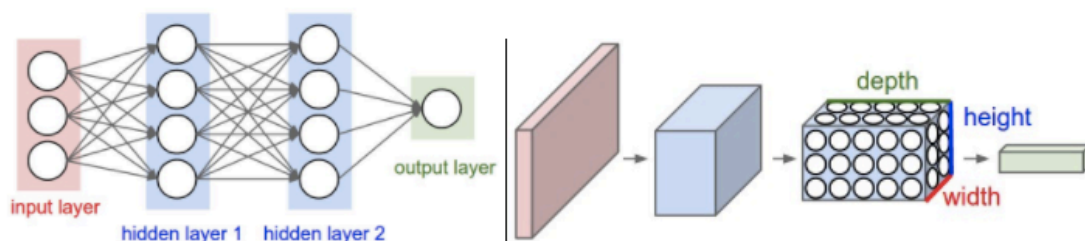


## Algorithms and Techniques

The classifier used for the given task is a Convolutional Neural Network which has achieved state of the art in recent times for image classification among many other things.

Convolutional Neural Networks are pretty similar to ordinary Neural Networks. They are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer. What differentiates them from ordinary Neural Networks is their explicit assumption about the inputs being images.

Unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. (Note that the word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network)



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Following groups of parameters need to be specified for training the network:

A. Training parameters:

1. Number of Epochs - Number of complete passes to perform over the entire dataset.
2. Batch Size - Number of training examples to feed to the model at one go.
3. Learning Rate - Crudely, the rate at which the model learns. A high learning rate means aggressive tuning of parameters and a low learning rate implies taking a more passive approach.














B. Neural Network Architecture:

1. Number of layers
2. Type of layers (Fully Connected Layers, Convolutional Layers etc.)
3. Layer parameters (Activation function, number of neurons, filter size etc.)

I also used data augmentation techniques to make the model more robust. Data augmentation techniques generate additional images from the original ones by performing transformations like rotating, mirroring etc. on the images of original dataset. This way, the model becomes invariant to changes in angle of faces etc., and is able to give better real world performance.

## Benchmark

The best accuracy on the test set was 0.71162 on Kaggle. Since I was limited by computational resources, my goal was to achieve at least 0.6 accuracy on the test set.

#	△pub	Team Name <small>* in the money</small>	Kernel	Team Members	Score ?	Entries
1	—	* RBM			0.71162	5
2	—	Unsupervised		 	0.69267	8
3	—	Maxim Milakov			0.68821	7
4	—	Radu+Marius+Cristi		  	0.67484	6
5	—	Lor.Voldy			0.65255	2
6	▲1	ryank			0.65088	2
7	▼1	Eric Cartman			0.64475	1
8	—	Xavier Bouthillier			0.64224	1
9	▲1	Alejandro Dubrovsky			0.63110	5
10	▼1	sayit			0.62190	2

## III. Methodology

### Data Preprocessing

The following preprocessing steps were performed:

1. The ExtractImages.py script extracts images from the CSV file into three different folders namely “Training”, “PublicTest” and “PrivateTest”.
2. Under the “Prepare Data” section in the ModelBuildingAnalysis notebook, the images as well as corresponding labels (from CSV file) into different arrays (train, valid, test). In this step, we also perform the following transformation on each pixel value (X) of the image.

$$X' = \frac{X - \text{Mean}(X)}{\text{Std}(X)}$$

where X' is the new pixel value.

3. I also created a separate generator for augmenting images using the following parameters:
  - rotation\_range - Degree range for random rotations. This is set as 40 degrees.
  - width\_shift\_range - Range of random horizontal shifts. This is set as 0.2.
  - height\_shift\_range - Range of random vertical shifts. This is also set as 0.2.
  - shear\_range - Shear Intensity (Shear angle in counter-clockwise direction as radians). This is also set as 0.2.
  - horizontal\_flip - Randomly flips input horizontally.

Setting of parameters for augmenting images is more of an art than an exact science. A lot of it depends on the domain and the inherent correlations present in training set. The exact parameters chosen by me were found by doing a lot of research over the internet and some intuition. I chose to only augment the images present in training set.





## Implementation

Once the data has been preprocessed and divided into training, validation and testing set; we train the classifier on the training set. The following steps are performed in the implementation stage:

### 1. Define the network architecture and set training parameters:

The final neural network architecture can be seen on the right. It has a total of 140,343 trainable parameters. The architecture was inspired from VGG net. I used Keras library with Tensorflow backend for implementing the model. The following types of layers are present in the model:

- Convolutional Layer:
- Batch Normalisation Layer:
- Max Pooling Layer:
- Dropout Layer:
- Dense (Fully Connected) Layer:

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 46, 46, 16)	160
batch_normalization_8 (Batch Normalization)	(None, 46, 46, 16)	64
conv2d_8 (Conv2D)	(None, 44, 44, 16)	2320
batch_normalization_9 (Batch Normalization)	(None, 44, 44, 16)	64
max_pooling2d_4 (MaxPooling2D)	(None, 22, 22, 16)	0
dropout_5 (Dropout)	(None, 22, 22, 16)	0
conv2d_9 (Conv2D)	(None, 20, 20, 32)	4640
batch_normalization_10 (Batch Normalization)	(None, 20, 20, 32)	128
conv2d_10 (Conv2D)	(None, 18, 18, 32)	9248
batch_normalization_11 (Batch Normalization)	(None, 18, 18, 32)	128
max_pooling2d_5 (MaxPooling2D)	(None, 9, 9, 32)	0
dropout_6 (Dropout)	(None, 9, 9, 32)	0
conv2d_11 (Conv2D)	(None, 7, 7, 64)	18496
batch_normalization_12 (Batch Normalization)	(None, 7, 7, 64)	256
conv2d_12 (Conv2D)	(None, 5, 5, 64)	36928
batch_normalization_13 (Batch Normalization)	(None, 5, 5, 64)	256
max_pooling2d_6 (MaxPooling2D)	(None, 2, 2, 64)	0
dropout_7 (Dropout)	(None, 2, 2, 64)	0
flatten_2 (Flatten)	(None, 256)	0
dense_3 (Dense)	(None, 256)	65792
batch_normalization_14 (Batch Normalization)	(None, 256)	1024
dropout_8 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 7)	1799
Total params: 141,303		
Trainable params: 140,343		
Non-trainable params: 960		

### 2. Define the loss function and optimiser:

As mentioned earlier, I used multi-class categorical cross entropy as the loss function for the model. Adam optimiser was used for tuning the parameters of the model.

### 3. Set training parameters:

The following training parameters were set:

- Number of epochs: It is very hard to speculate about the number of epochs required by the model. I adopted the strategy of early stopping, i.e., stopped training the model when the validation loss stopped decreasing or when the model began to overfit the training set.
- Learning Rate: The default parameters set by the optimiser were used.
- Batch size: The batch size was set as 128.

4. **Training:** Once the model has been defined and compiled, the next step is to feed training data to it. Since data augmentation is computationally expensive, I chose to first train the model on the original training data, i.e., without feeding any augmented images and measure the performance. I then trained a new model, feeding it augmenting images as well this time and measured the performance.

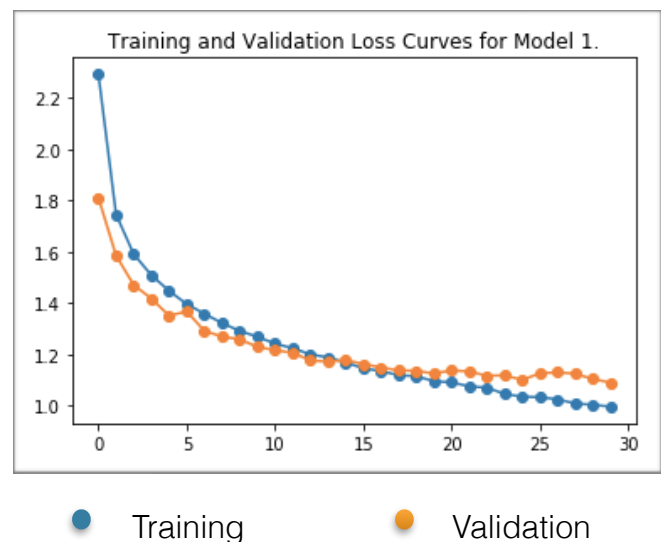
## IV. Results

### Model Evaluation and Validation

Following are the training loss and validation loss observed for the 2 different models:

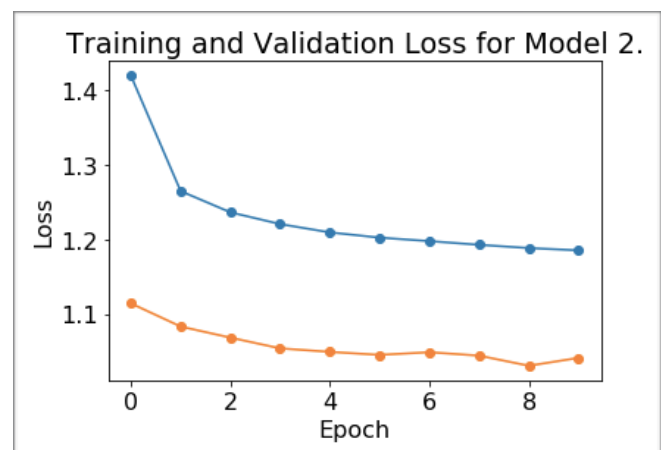
- As mentioned earlier, Model 1 was trained on the original training set and no augmented images were fed to it. From the figure on the right, it becomes that the model has fit the data well and is at the verge of starting to overfit. Thus, it was stopped after the 30th epoch.

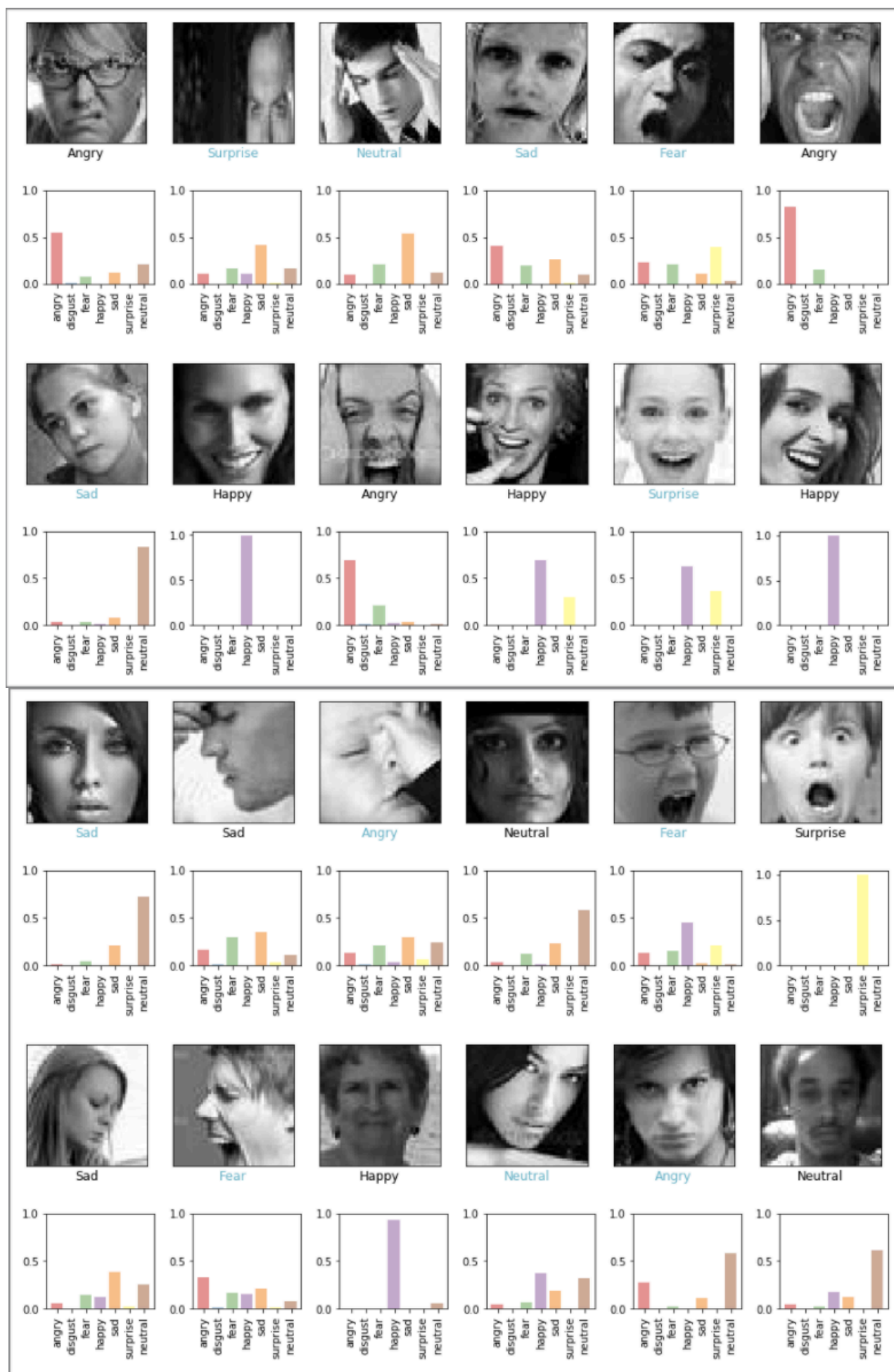
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
10	0.55246	1.18527	0.61204	1.04106



- In contrast to Model 1, we trained Model 2 using original images as well as augmented images. From the figure on the right, we can see that the model has currently underfit the data and there's a lot of scope for improvement in the model. However, each epoch was taking 2 hours to complete and as a result, I decided to stop training the model after 10 epochs.

Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
30	0.62684	0.99654	0.59281	1.09049



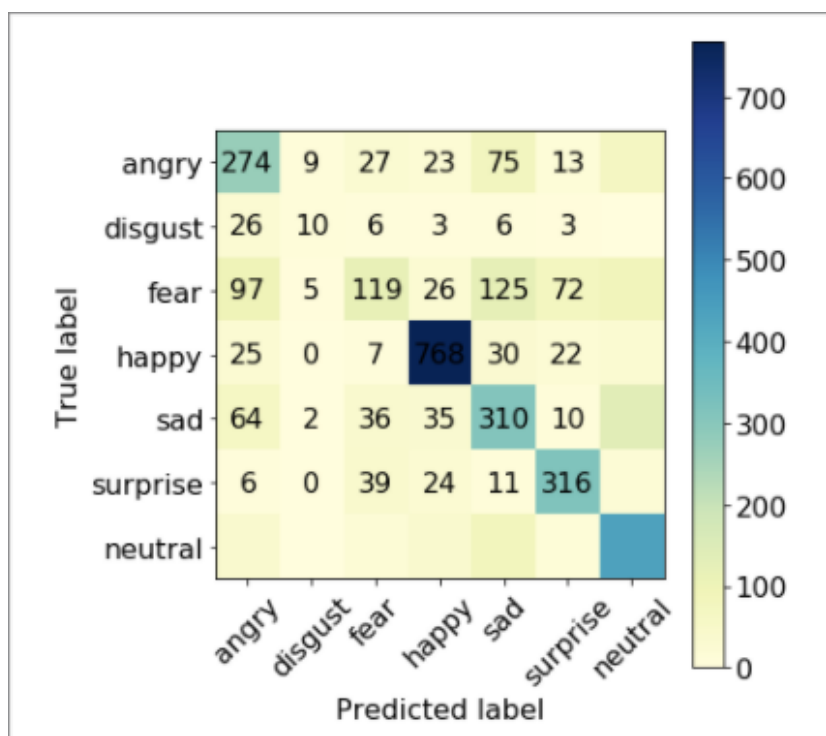


The test set performance of the two models is as follows:

Model	Test Accuracy	Test Loss
1	0.58902	1.06349
2	0.62162	1.00026

Thus, by using data augmentation I was able to increase accuracy by almost 3 percent. Also, to reiterate, the model is still undercutting the data and accuracy would definitely been better had we trained the model for more epochs.

Below is the confusion matrix obtained from the test set for the various emotions:



The model seems to be confusing the following pairs of emotions:

- Anger and Sad
- Disgust and Anger
- Fear and Sad
- Fear and Surprise

Intuitively, it makes sense as barring Anger and Sad, the above pairs of emotions can have similar expressions. For example, an expression denoting fear can both have mouth and eyes wide open (similar to surprise) or a much more passive expression (similar to sad). Thus, this is not an inherent defect with the model, but the ambiguity of human expressions which makes it hard to differentiate.

From the pictures above, it becomes obvious that a lot of images that the model classifies incorrectly are hard even for humans. Also, in many cases, the correct emotion is the one with the second highest probability.

Below are the precision, recall, f1 score and support metrics:

- precision = 
$$\frac{TruePositives}{TruePositive + FalsePositives}$$

The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.

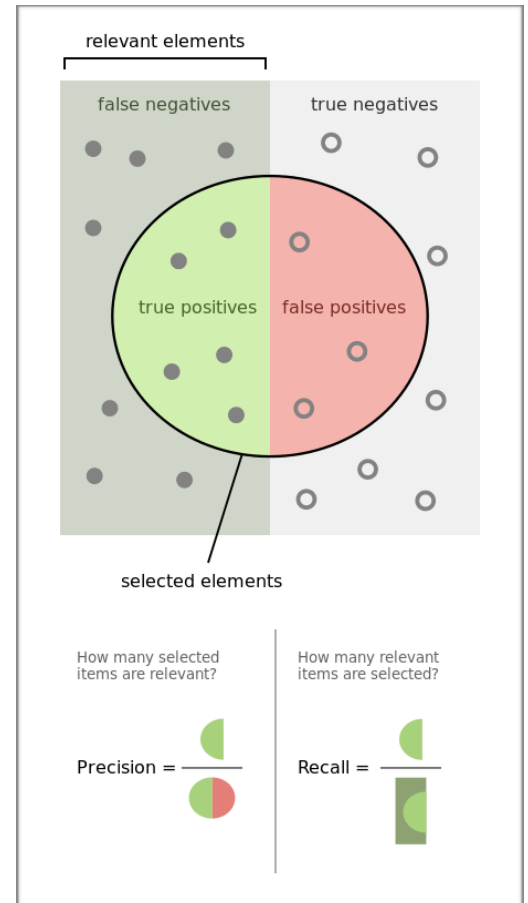
- recall = 
$$\frac{TruePositives}{TruePositives + FalseNegatives}$$

The recall is intuitively the ability of the classifier to find all the positive samples.

- f1 score = 
$$2 * \frac{precision * recall}{precision + recall}$$

The F-beta score can be interpreted as a weighted harmonic mean of the precision and recall, where an F-beta score reaches its best value at 1 and worst score at 0.

- support - It is the number of occurrences of each class in the actual labels or y\_true.



In the table below, we have the above metric calculated on the test set. The model has learned to identify happy and surprise emotion the most accurately or with the highest f1 scores and disgust and fear the most poorly or with the lowest f1 scores.

Disgust makes sense, because we had very few training examples for it and thus, the model wouldn't have been able to generalise. Poor performance on the emotion fear can be attributed to the ambiguity factor in human expressions.

	precision	recall	f1-score	support
angry	0.52	0.56	0.54	491
disgust	0.37	0.18	0.24	55
fear	0.47	0.23	0.30	528
happy	0.84	0.87	0.86	879
sad	0.48	0.52	0.50	594
surprise	0.70	0.76	0.73	416
neutral	0.56	0.69	0.62	626
avg / total	0.61	0.62	0.61	3589

## V. Conclusion

### Final Thoughts

Thus, I was able to train a ConvNet for identifying the emotion based on static images of human faces. I also showed that using Data Augmentation techniques can improve the model by almost 3% in the case of a domain where you have noisy images with inconsistent lighting, poses etc.

The accuracy achieved by the final model on the test set was 62.162% which would have almost made it to the top 10 of Kaggle Leaderboard. It is still far from the state-of-the-art 71.162%. However, I'm hopeful that with a little more of data cleaning, hyper parameter tuning and a more complex architecture, I would have atleast achieved more than 65% accuracy.

### Improvement

There is a lot of scope for improvement in the above model. I was limited by computational resources and as a result couldn't experiment much with different architectures or hyper parameter tuning. Also, the data was very noisy and more sophisticated data cleaning steps would have Definitely increased the accuracy of the model.

As seen in data exploration section, the number of examples for "Disgust" emotion are very low. As a result, we obtain very poor performance for this emotion. In order to improve this, we can either merge "Disgust" and "Anger" into one category or go out and collect more images for "Disgust".

In practice, very few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pretrain a ConvNet on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the ConvNet either as an initialization or a fixed feature extractor for the task of interest. I would have like to explore this approach as well and see how it compares to training the model from scratch.

## References:

- [1] <https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data>
- [2] <http://machinelearningmastery.com/image-augmentation-deep-learning-keras/>
- [3] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. JMLR Proceedings, 2015.
- [4] F. Chollet. keras. <https://github.com/fchollet/keras>, 2015.
- [5] [https://en.wikipedia.org/wiki/Precision\\_and\\_recall](https://en.wikipedia.org/wiki/Precision_and_recall)
- [6] <http://cs231n.github.io/transfer-learning/>
- [7] <https://github.com/JostineHo/mememoji>
- [8] [http://cs231n.stanford.edu/reports/2016/pdfs/005\\_Report.pdf](http://cs231n.stanford.edu/reports/2016/pdfs/005_Report.pdf)
- [9] <http://cs231n.github.io/convolutional-networks/>