# CSE 527: Introduction to Computer Vision
# Semester Project Report

# Facial Beauty Detection using Convolutional Neural Network

Compiled by Aaswad Satpute (SBU ID: 109955001) &
Chetan Naik (SBU ID: 109930376)
Under the guidance of Prof. Dimitris Samaras

At Computer Science Department,
Stony Brook University,
NY

# Introduction

This is an implementation of the research paper, Douglas Gray, Kai Yu, Wei Xu, and Yihong Gong. 2010. Predicting facial beauty without landmarks. *In Proceedings of the 11th European conference on Computer vision: Part VI (ECCV' 10)*, Kostas Daniilidis, Petros Maragos, and Nikos Paragios (Eds.). Springer-Verlag, Berlin, Heidelberg, 434-447.

We have taken the image dataset of faces from the Stony Brook vision lab for training and testing purposes. Aim of this project is, for every image with a face of a human, rate the beauty of the face, using a Convolutional Neural Network (CNN). This is done without concedering any landmarks. Thus, detecting the features, training them, as well as testing would be done using CNN.

# Dataset

For training and testing we used image dataset of faces from the Stony Brook vision lab. This dataset is available here: vision.cs.stonybrook.edu/~lehhou/mixture.zip

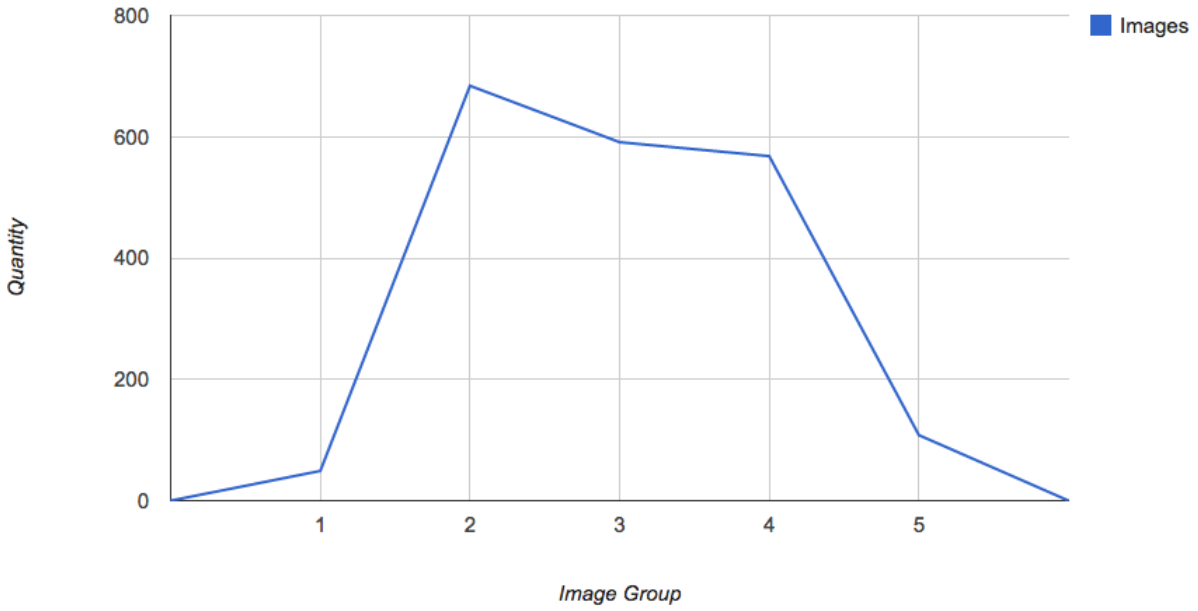
We had also planned to use the image dataset of University of Massachusetts at Amherst called labelled faces in the wild database. But due to time constraints we could not integrate the dataset in our project. This dataset is available here: http://vis-www.cs.umass.edu/lfw/

# Our Approach

We gather the data from Stony Brook Vision Lab. Once we have all the data with us we then start with our process as per the paper.

We unarchive the 2000 images. We segregate these images into 5 rankings viz. 1 to 5. These are unevenly distributed, because these were manually segregated. We have rated 49 images as in 1st group, 684 images as in 2nd group, 591 images as in 3rd group, 568 images as in 4th group, and remaining 108 images as in 5th group.

This is illustrated in the graph below.




Once this was done, we started with constructing the Convolutional Neural Networks or the CNN. For this we tried to install caffe on our systems. But unfortunately because both the systems we have, were already upgraded to Mac OS X 10.10 Yosemite. Yosemite just being released this fall, has not yet been supported by caffe. This broke our system and we had to rebuilt it.

So we decided to install caffe on a virtual machine. We installed a virtual machine (VM) management system called Vagrant. We used a Vagrant box having a flavor of Ubuntu which has Caffe preinstalled on it. Vagrant can be found on this link here: https://www.vagrantup.com/. We then learned how the CNN works in caffe. We then first ran small CNNs inside caffe. We started training using ImageNet on one system and LeNet on other.

Once we were acquainted with how this works, we then started running our images to train the CNN. We edited the configuration files which described the structure of the neural network. We have trained using 1400 images and the testing dataset consists of remaining 600 images.

This required a lot of trial and error. We had to run and rerun the system many a times. The accuracy kept on fluctuating. We ran this with various depth, batch sizes and different structure of the network. And also varied our learning rates too. We then randomized the image set. This also varied the accuracy of our model.

To summarize, we experimented various configurations by changing the following things.

- Layer parameters
    - number of outputs of a layer
    - kernel size
    - stride
- Solver parameters
    - learning rate
- Layer structure
    - add/delete CONVOLUTION layer
    - add/delete POOLING layer
    - add/delete RELU layer
    - add/delete LRN layer

Then we found the following configuration to give us most accurate results.

- Layer Parameters:

```
[('data', (10, 3, 120, 90)),
 ('conv1', (10, 50, 28, 20)),
 ('pool1', (10, 50, 14, 10)),
 ('conv2', (10, 100, 10, 6)),
 ('pool2', (10, 100, 5, 3)),
 ('ip1', (10, 500, 1, 1)),
 ('ip2', (10, 5, 1, 1)),
 ('prob', (10, 5, 1, 1))]
```

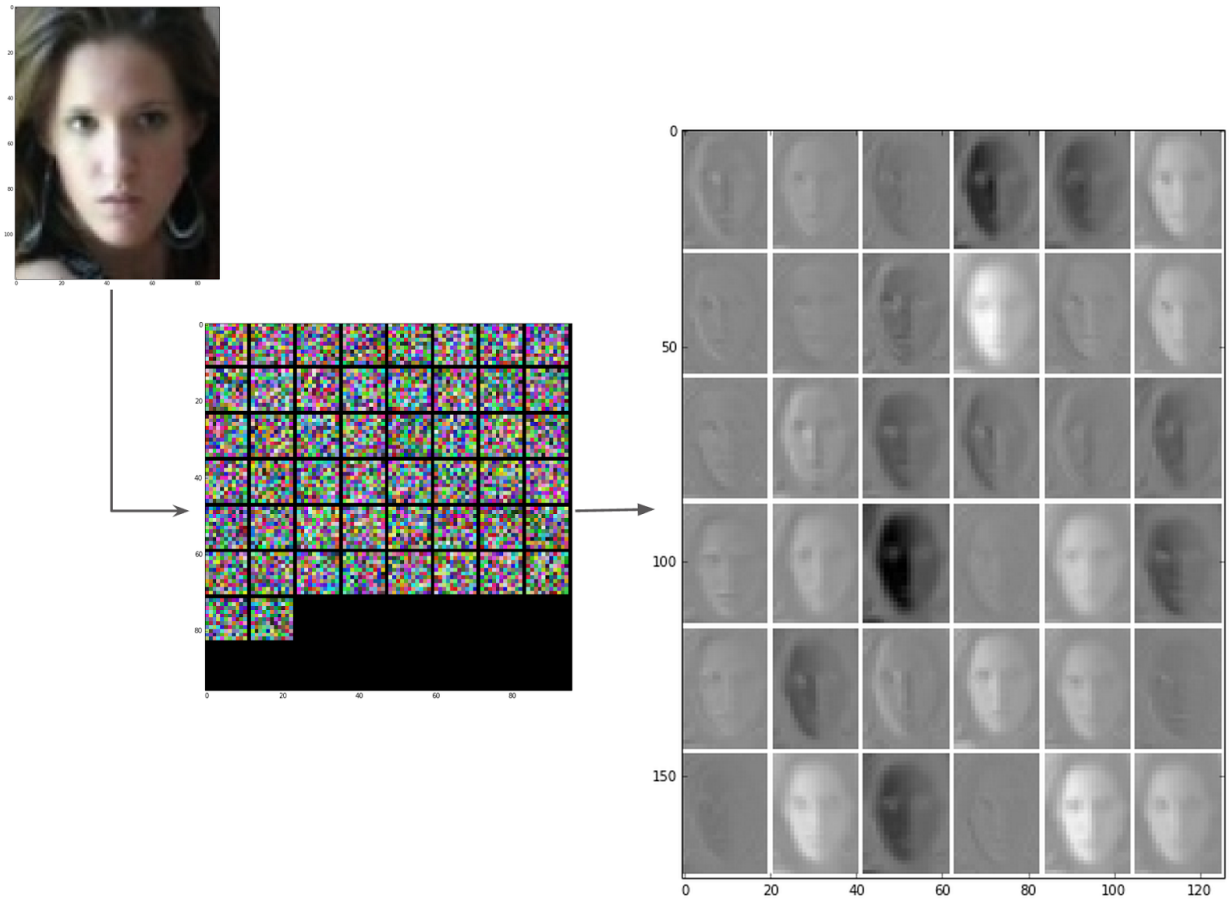
  Here each has the data in following key value pair format: the first part is the name of the layer, and the other part has batch size and training size. For example: in the first layer “data” is the name of the layer because it contains input data. Then “10” is the batch size for which the CNN would create a batch as input. The “3” following it is for 3 channels namely Red, Green, and Blue. And similarly the last layer called “prob” is output layer. Here “10” is again the batch size. Then “5” is the number of nodes because we have 5 different groups. Then last two parameters are “1” each because the output only has one value, and that is the probability of being considered in that group.

- Solver Configuration:

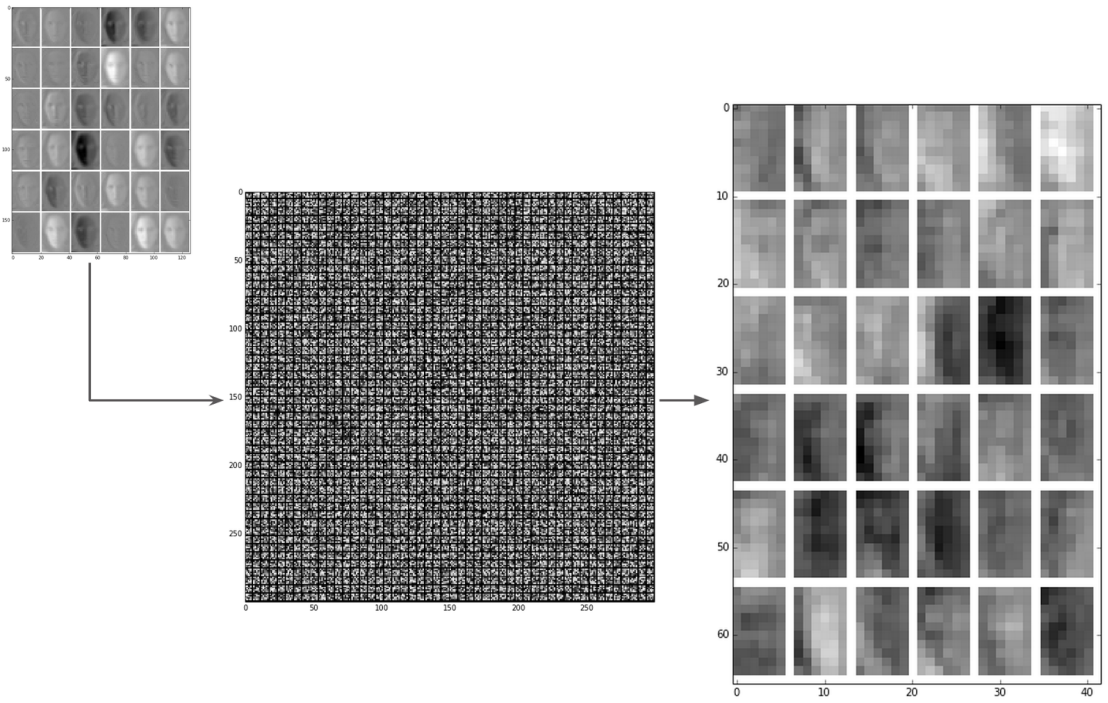
```
test_iter: 100
test_interval: 100
base_lr: 0.001
momentum: 0.9
weight_decay: 0.0005
lr_policy: "inv"
gamma: 0.0001
```

```
power: 0.75
display: 100
max_iter: 5000
snapshot: 500
snapshot_prefix: "fbeauty"
solver_mode: CPU
```

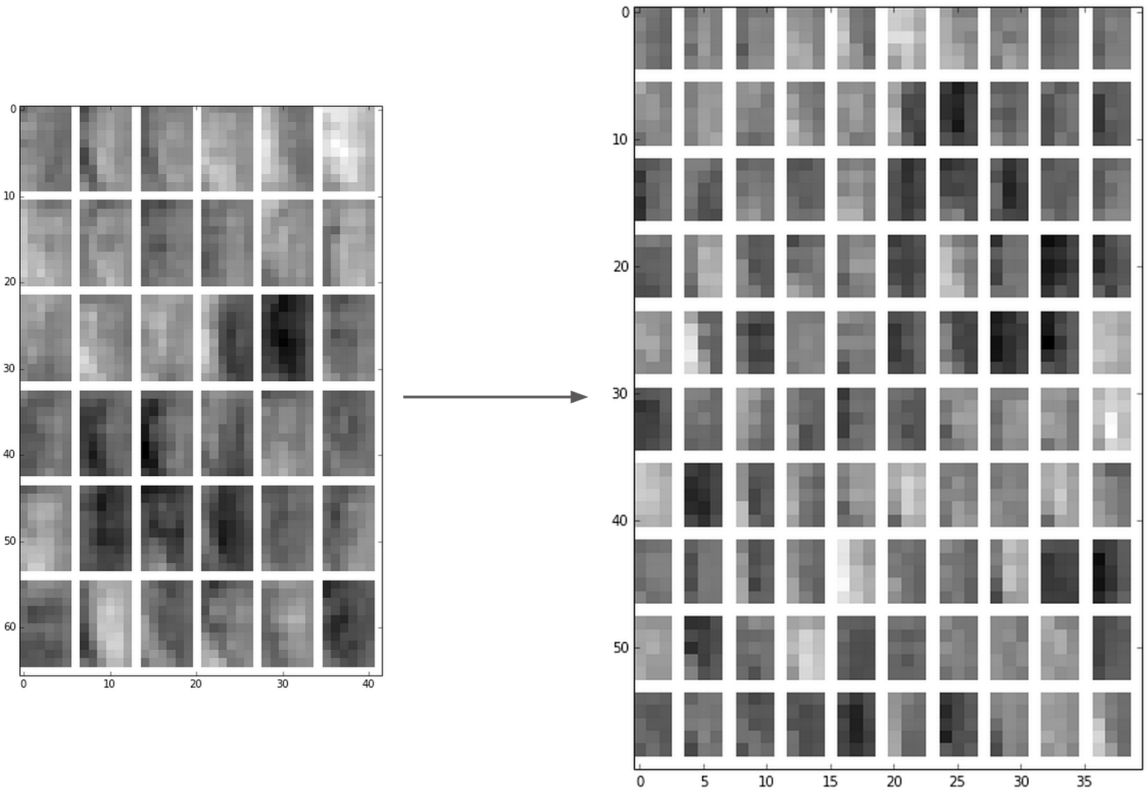
Now let's walk through one example. First one image is taken as input. Then this image is given into the first convolutional layer with 50 nodes. The first layer gives the output to the next layer as shown below.



This output then is given as input to the next convolutional layer. As follows:



And we perform pooling on the output of convolutional layer as shown below.

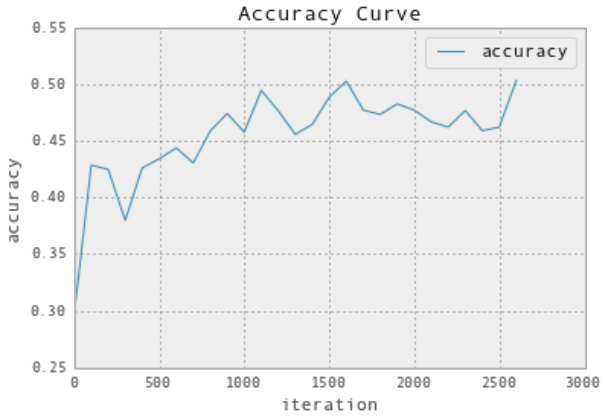


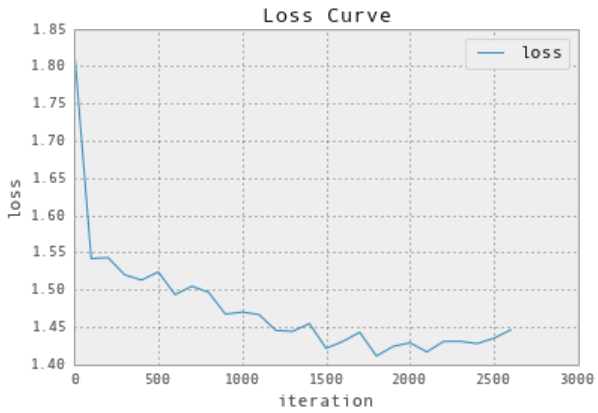
The outputs from the convolutional layers are then passed to the INNER_PRODUCT later which is a fully connected neural network. There are two such layers in our CNN and this structure will output 5 classes. We then finally use a SOFT_MAX LOSS layer to get the class probabilities.

The various accuracies and the results have been discussed in the next section.

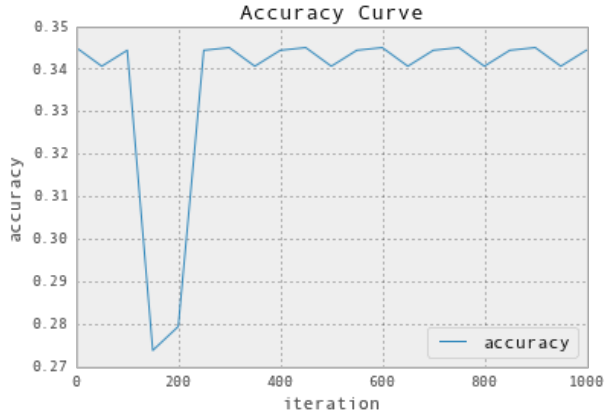
# Results

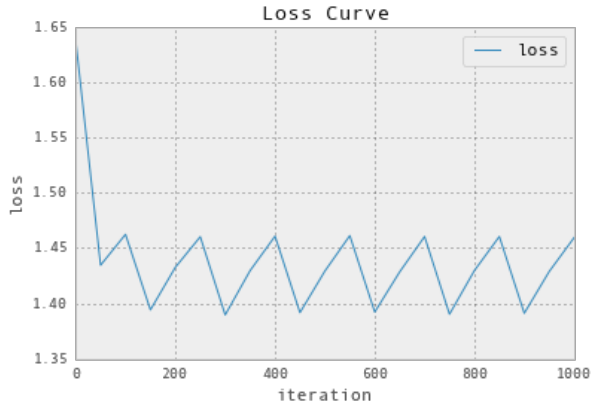
The model described above achieved around 50.32% accuracy after about 2600 iterations. The following plots show how the accuracy and the loss of the model changes as the training iterations were increased.

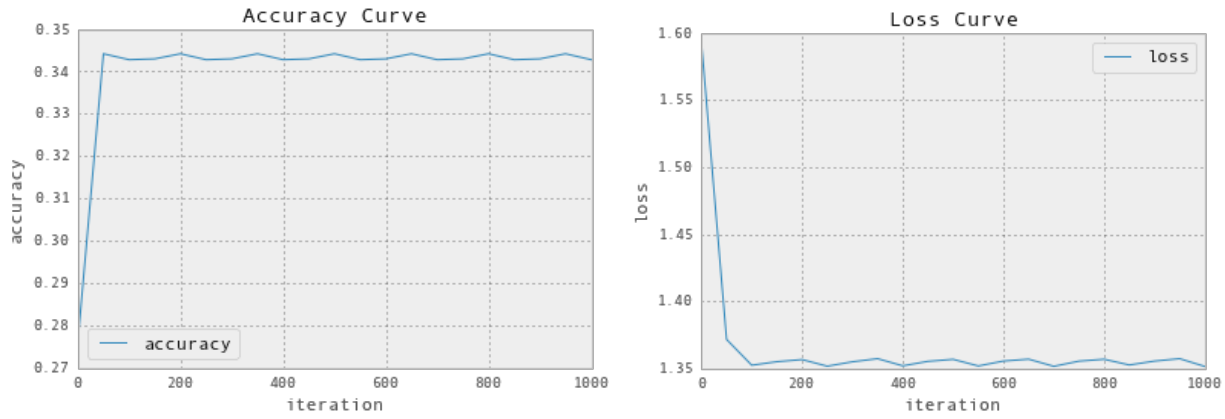



The following plots show accuracy and loss curves of initial experimental models. You can see that these have slightly lower accuracy. Also, the accuracy and loss in one of the experiments had sawtooth pattern. We believe that this happened because of big learning rate parameter.






We now show a small part of our result data. Now here, name means the name of image. Rating is the human rating given to the same image. predicted_rating is the rating predicted by our model. probability is the probability of the predicted rating being true as per our model. error column shows the error in prediction calculated using `error = actual rating - predicted rating`. The following table shows result for training dataset.

| | name | rating | predicted_rating | probability | error |
|---|---|---|---|---|---|
| **0** | 774.jpg | 2 | 2 | 1.000000 | 0 |
| **1** | 471.jpg | 3 | 4 | 1.000000 | -1 |
| **2** | 1240.jpg | 4 | 4 | 1.000000 | 0 |
| **3** | 1680.jpg | 2 | 4 | 1.000000 | -2 |
| **4** | 639.jpg | 2 | 4 | 1.000000 | -2 |
| **5** | 603.jpg | 3 | 4 | 1.000000 | -1 |
| **6** | 1467.jpg | 3 | 3 | 1.000000 | 0 |
| **7** | 1748.jpg | 4 | 4 | 0.986405 | 0 |
| **8** | 347.jpg | 2 | 3 | 1.000000 | -1 |
| **9** | 376.jpg | 4 | 4 | 1.000000 | 0 |
| **10** | 795.jpg | 4 | 3 | 1.000000 | 1 |
| **11** | 94.jpg | 3 | 3 | 1.000000 | 0 |
| **12** | 186.jpg | 4 | 4 | 1.000000 | 0 |
| **13** | 1919.jpg | 2 | 3 | 1.000000 | -1 |
| **14** | 953.jpg | 4 | 2 | 1.000000 | 2 |

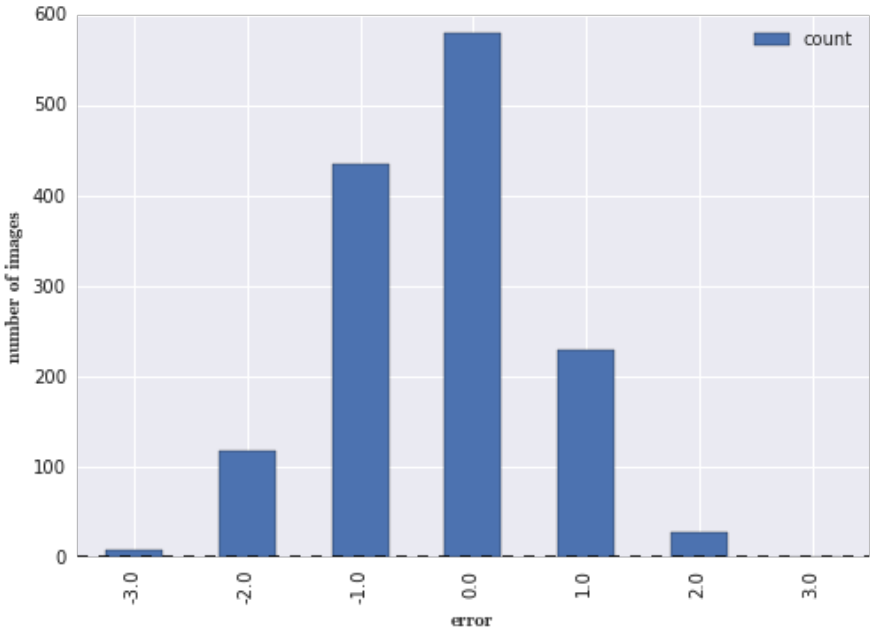
Similarly the following table shows the results for testing dataset.

| | name | rating | predicted_rating | probability | error |
|---|---|---|---|---|---|
| **0** | 1106.jpg | 3 | 3 | 0.999996 | 0 |
| **1** | 1459.jpg | 2 | 3 | 1.000000 | -1 |
| **2** | 293.jpg | 3 | 3 | 1.000000 | 0 |
| **3** | 402.jpg | 4 | 4 | 1.000000 | 0 |
| **4** | 532.jpg | 4 | 3 | 1.000000 | 1 |
| **5** | 617.jpg | 4 | 3 | 1.000000 | 1 |
| **6** | 13.jpg | 3 | 3 | 1.000000 | 0 |
| **7** | 325.jpg | 4 | 3 | 1.000000 | 1 |
| **8** | 969.jpg | 3 | 3 | 1.000000 | 0 |
| **9** | 262.jpg | 4 | 4 | 1.000000 | 0 |
| **10** | 1137.jpg | 4 | 4 | 1.000000 | 0 |
| **11** | 111.jpg | 4 | 3 | 0.500000 | 1 |
| **12** | 1823.jpg | 2 | 3 | 0.572685 | -1 |
| **13** | 1897.jpg | 2 | 3 | 1.000000 | -1 |
| **14** | 1525.jpg | 5 | 3 | 1.000000 | 2 |

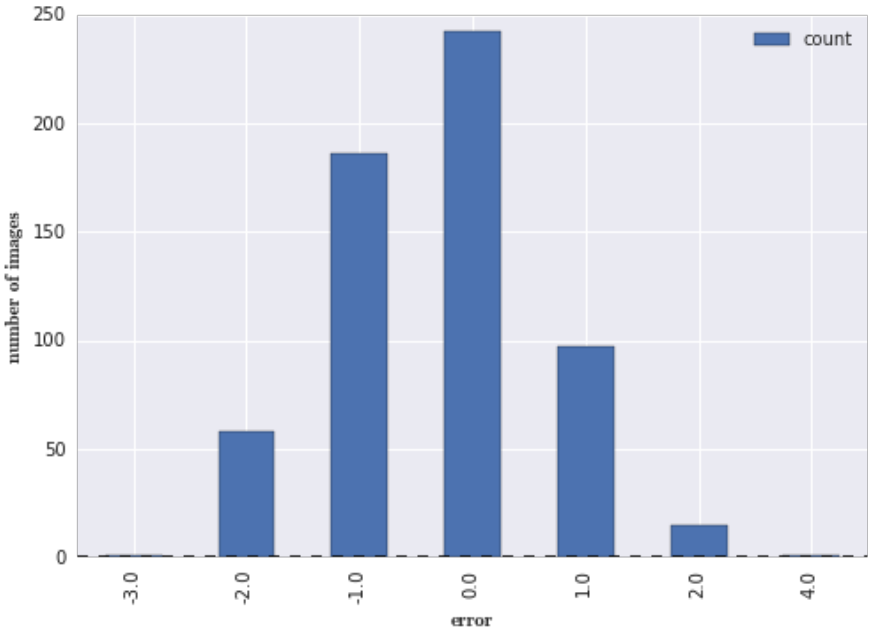
As you can see, the model is predicting image rating with ~50% accuracy. We also calculated the correlation between actual and predicted rating for the testing and training images. The training correlation value is 0.3934. And the testing correlation value is 0.3772. To see how much the prediction deviated exactly from the human rating, we have plotted the histogram of errors in both training set as well as test set.

This can also be used to infer if the model is overfitting. You can notice that the error distribution of both test set and train set look alike. Thus we can conclude that the model is not biased towards train set. Also you can notice that about predicted_rating of ~50% of the images are zero. And, about 45% image rating predictions are off by just one value either in the positive or negative direction. Only about 5% of the predicted_ratings are highly deviant (>2 deviation).

The following graph shows the error of output of our model on the training image dataset.




Similarly the following graph shows the error of output of our model on the testing image dataset.



# Conclusion

Training time increased with two factors, number of convolutional layers, and number of outputs of a layer. If the network configuration is set up correctly based on theoretical knowledge and empirical analysis, as the number of iterations increase, accuracy increases. When we randomized the train set differently, we got different accuracies, even though we used the same network. This shows that having right training set along with having more number of training images decides the accuracy of the CNN. So, we can conclude that the results could be improved if more images are used for training and more number of training iterations are performed using a system with dedicated GPU.

# Resources

We have done the project in Python. We used open source implementation of CNN called Caffe (Deep learning framework developed by Berkeley Vision and Learning Center). For training and testing we used image dataset of faces from the Stony Brook vision lab.

# Bibliography

1. Caffe - http://caffe.berkeleyvision.org/
2. Vargrant - https://www.vagrantup.com/
3. Dataset from Stony Brook University - vision.cs.stonybrook.edu/~lehhou/mixture.zip
4. University of Massachusetts at Amherst's labelled faces in the wild database - http://vis-www.cs.umass.edu/lfw/
5. Douglas Gray, Kai Yu, Wei Xu, and Yihong Gong. 2010. Predicting facial beauty without landmarks. *In Proceedings of the 11th European conference on Computer vision: Part VI (ECCV' 10)*, Kostas Daniilidis, Petros Maragos, and Nikos Paragios (Eds.). Springer-Verlag, Berlin, Heidelberg, 434-447.
6. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE* , vol.86, no.11, pp.2278,2324, Nov 1998.
7. OpenCV - http://opencv.org/