Facial Keypoint Detection

CS365 – Artificial Intelligence

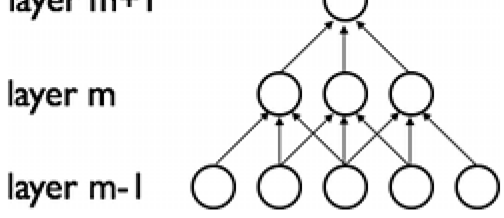
Abheet Aggarwal 12012 **Ajay Sharma 12055**

Abstract

Recognizing faces is a very challenging problem in the field of image processing. The techniques presently being used are Biometric recognition (but Iris scanners are far too expensive), Eigenfaces(inaccurate with varying image factors like intensity, camera angles), line edgemaps. While facial features in images depend on a lot many conditions , faces can be recognised easily if we use the relevant keypoints and landmarks for identification. The unchanging ratios and distances between these mark the importance of this approach.

Introduction

The challenge is to predict keypoint positions on face images. Facial features vary greatly for an individual as we have different poses , angles and even the different lighting conditions also make it difficult to predict . We have tried to solve this problem in two ways: using Neural Network(NN) , and using Convolutional Neural Network(CNN). In NN, all the nodes in neighbouring layers are fully connected, while in CNN, they are locally connected(i.e. the inputs of hidden units in layer *m* are from a subset of units in layer *m-1)*.



Img source:www.deeplearning.net

For NN , we took a structure with just one hidden layer having 100 nodes, whereas for CNN, the structure consists of a three-level cascade of convolutional layers each followed by a pooling layer. The network ends into two fully connected hidden layers(they have been chosen 2 so as to reduce the computation time since decreasing the number of hidden layers exponentially increases the time). CNN is a type of feed forward neural network which locally shares weights. At the end we come up with some adjustments in the same network to make it more accurate and save time.

Motivation

Detection of facial keypoints is building block for many applications in Computer Vision. Research has been done on this part but still there is hope for improvement. What really motivated us was that this problem works as the first step for many applications like :

1. for detecting faces in an image or even in a video
2. for analysing facial expressions
3. automatically finding the desired shape faces (like the golden ratio faces) among a huge collection
4. face recognition that works with changing ages of people

Use of deep learning made us more interested in the project. We have chosen Convolutional Neural Network because it is more accurate and allows us to modify at various stages. Some exciting work has been done in the field of facial keypoint detection by Yi Sun and others.Ref[1]. It really motivated to use Convolutional Neural Network for the problem. And last but the video that motivated us the most is this :
https://www.youtube.com/watch?v=xpBXpI39s9c .
After watching this we thought let's give it a try and try to do what he did so well.

Dataset

Data was taken from the site www.kaggle.com . It has 7049 images in training dataset out of which only 2140 images had all the 15 keypoints labelled . All the images are grayscale images and of size 96X96 pixels where value of each pixel is given in the range 0-255 . All the training data is in the form of a csv file in which each row contains all the information regarding one image including the value of each pixel and the labelled keypoints coordinates. We use 80 % of the training data for training and other 20 % as validation data. The accuracy of the structure is checked by the MSE on this validation data.

Previous Works

The approach of using eigenfaces for recognition was developed by Sirovich and Kirby (1987) and used by Matthew Turk and Alex Pentland in face classification. In 1996 Second IEEE conference, R Herpers came out with an approach[Ref-5] for detecting facial features and characteristic anatomical keypoints. It was based upon extracting edge information from faces and exploit it to search and identify faces. For example, quoting the paper itself "The most prominent and reliable features within the eye region are the edges of the iris. Therefore, the sequential search starts by detecting the left edge segment (vertical, bright-to-dark step edge) of the iris applying the basis filter operation. The detected edge is tracked upwards and downwards using another filter operation until the intersection points with the eyelids are reached". Clearly the method needed a precise edge and line detector which could be flexible at the same time and they used special filters which were steerable. Later people tried to use PCA approach to extract feature vectors from images but that too failed in conditions of varying lighting and changing poses of the same object.

The deep learning based neural networks approach came much later (around 2005) and has shown outstanding results. Also that rapid improvements in the field have boosted the applications of this into various fields. An article on technologyreview.com[Ref-6] points out that this paradigm, based on monkey brains, is close to how humans interpret and recognize faces. Sun, Y., Wang, X., & Tang, X. in 2013 gave a related approach in their paper "Deep convolutional network cascade for facial point detection"[Ref-1] that used three-level cascaded convolutional neural network to deal with the problem. They extracted only high-level features over the whole face at the initialization stage, which helps to locate high accuracy keypoints. In this way, the texture context information over the entire face is utilized to locate each keypoint. Also the geometric ratios and constraints among the keypoints are taken care of as the networks are trained to predict all the keypoints simultaneously.

Our Work (till Poster Presentation)

We tried to solve the problem using two different ways, first by using Simple Neural Network and then by using Convolutional Neural Network.

For the first part we made a structure including input layer of images 96X96 size , a hidden layer having 100 nodes and then an output layer which had 30 outputs(x and y coordinates of the 15 keypoints). We trained the structure for 100 epochs over the training data and calculated the loss on validation set.

In the second part i.e. Convolutional Neural Network, the architecture includes an input layer of size 96X96 , then a convolutional layer having 32 filters of size 3X3 followed by a pooling layer of 32 filters of size 2X2 each. After pooling we get an output of size 47X47X32 then we continue this structure with two other layers with smaller filter size as shown below in the Fig 1 and the number of outputs from each layer are shown in Fig 2. The Architecture also contains 2 hidden layers after the 3rd pooling layer each having 100 nodes and then comes the output layer which has 30 output nodes(x and y coordinates of the 15 keypoints). We trained the CNN structure for 100 epochs. The data at the input layer has been normalized (coordinates (0,96) to (-1,1) and the pixel values (0-255) to (0-1)) . The non-linearity function that we used in the hidden layers is the rectifier function. We used it over sigmoidal (tanh) function because it helps overcome the vanishing gradient problem[Ref-2]. All the help regarding coding in python to design neural networks using Lasagne and theano libraries was taken from Daniel Nouri's blog[Ref-3].

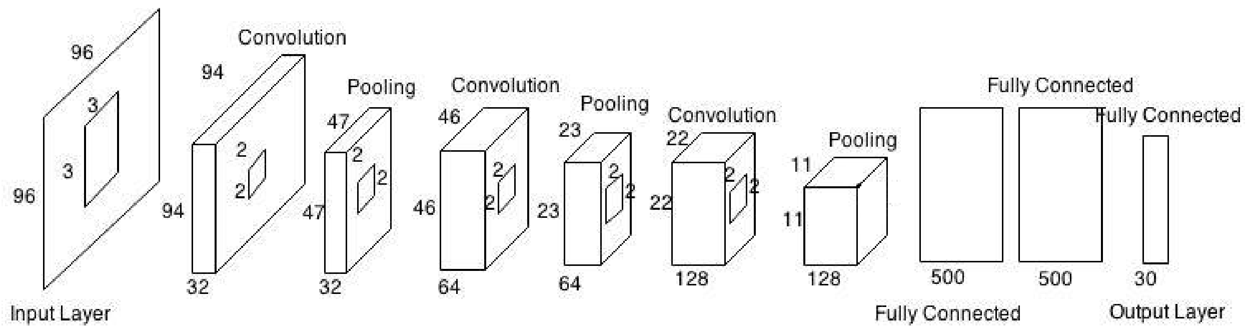
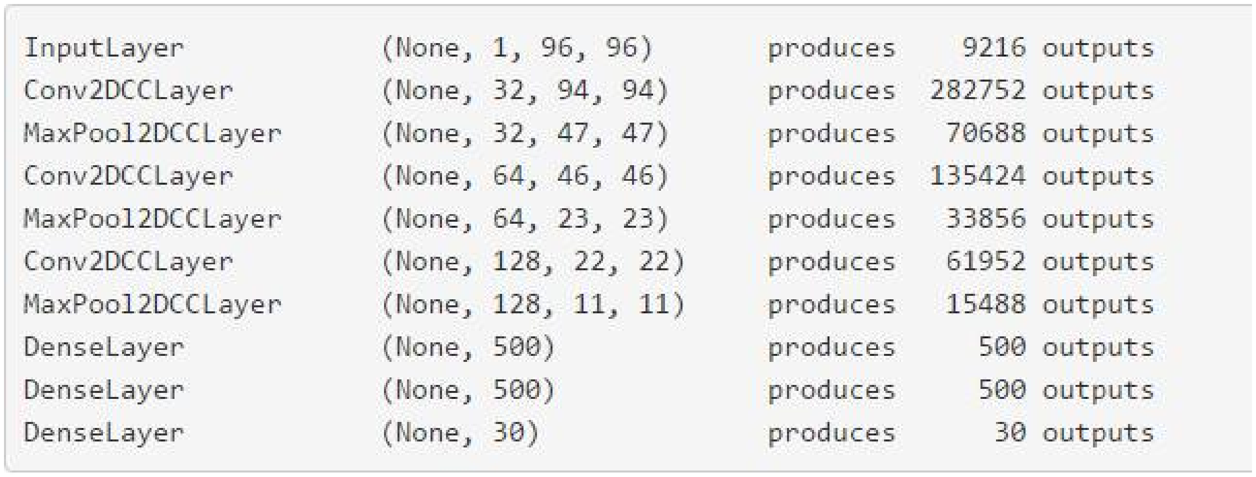


Fig 1. Architecture of Convolutional Neural Network



```
InputLayer            (None, 1, 96, 96)       produces    9216 outputs
Conv2DCCLayer         (None, 32, 94, 94)      produces  282752 outputs
MaxPool2DCCLayer      (None, 32, 47, 47)      produces   70688 outputs
Conv2DCCLayer         (None, 64, 46, 46)      produces  135424 outputs
MaxPool2DCCLayer      (None, 64, 23, 23)      produces   33856 outputs
Conv2DCCLayer         (None, 128, 22, 22)     produces   61952 outputs
MaxPool2DCCLayer      (None, 128, 11, 11)     produces   15488 outputs
DenseLayer            (None, 500)             produces     500 outputs
DenseLayer            (None, 500)             produces     500 outputs
DenseLayer            (None, 30)              produces      30 outputs
```

Fig 2 : Layer wise input and output dimensions

Now we have set the validation error to be default type which is MSE and as we have divided each target coordinate by 48 when we scaled them to be in [-1,1],so to find RMSE we use :

RMSE = $\sqrt{MSE} * 48$ ……..(1)

Simple NN has RMSE=3.8251 more than CNN(3.3014). Now , we got that we can achieve better results with CNN so we further try to make CNN better.

To further optimise our solution we can try to increase the hidden layer nodes because the more the nodes, better will be the results[Ref-8] only that it increases the computation time. So we now set the number of nodes in hidden layers to 500 and train for 100 epochs only. We get the new RMSE to be better than the earlier model where we had put 100 nodes.

Results so far

The results we got are in the form of loss on validation dataset(MSE) every time the net completes the process. We compute the RMSE as shown in Eqn.1 on last page. The average time required for an epoch to complete was 10 minutes. On a powerful GPU this could be reduced.

Simple Neural Network:

Training epochs	Validation loss	Hidden layer units	RMSE
50	0.006338	100	3.8251

CNN:

Training epochs	Validation loss	Hidden layer units	RMSE
100	0.004730	100	3.3014
100	0.004194	500	3.1085

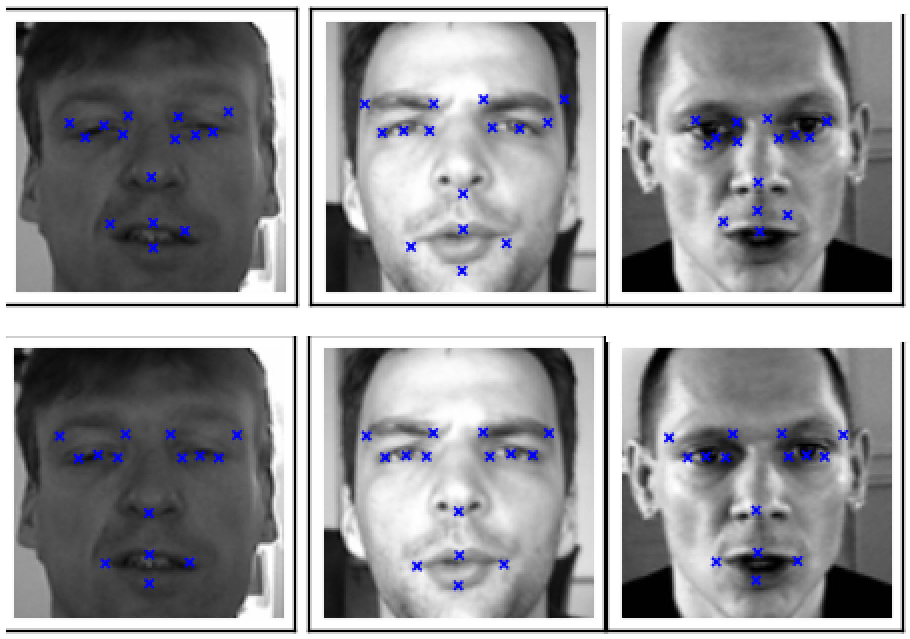


Fig 3 : Comparing both models on three test cases with unusual facial gestures clearly depicts how well a CNN(below) performs over Simple NN (specially around eyes and mouth).

Improvement in previous model

After poster presentation we tried to try something new. We tried to change the filter size for the convolutional layers.
While surfing over net, in answer to a question on stackoverflow.com[Ref-7], we found that for large size of the image like 96X96 if we take big filter size, it will give us good results as then less information will be lost. So we tried to make a new architecture with change of filters in convolutional layer as can be seen below in Fig 4 and the new no. of outputs after each layer in Fig 5.

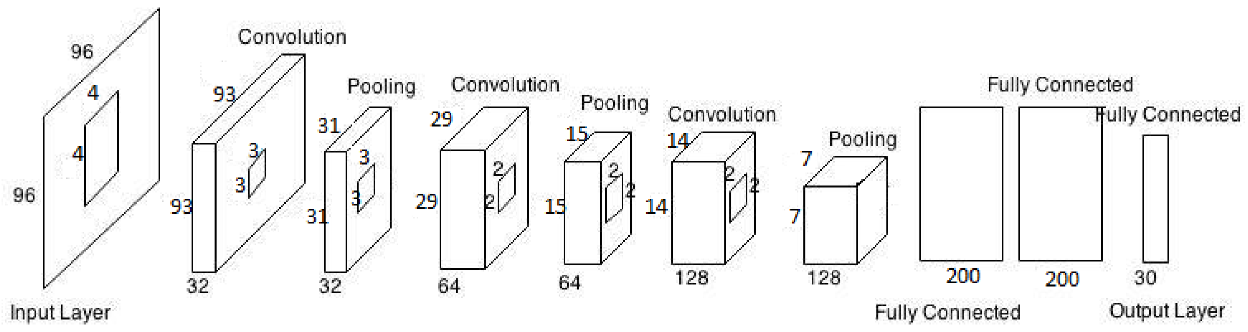
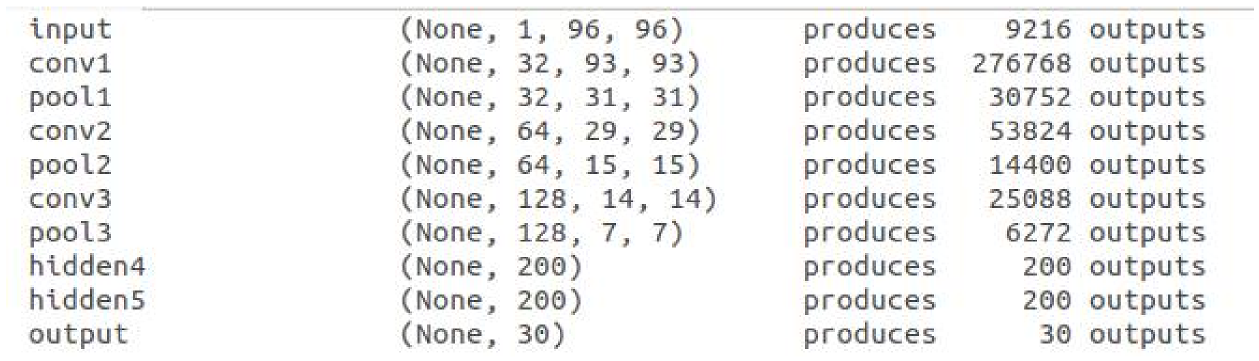


Fig 4: Architecture of Improved Convolutional Neural Network



```
input                (None, 1, 96, 96)       produces    9216 outputs
conv1                (None, 32, 93, 93)      produces  276768 outputs
pool1                (None, 32, 31, 31)      produces   30752 outputs
conv2                (None, 64, 29, 29)      produces   53824 outputs
pool2                (None, 64, 15, 15)      produces   14400 outputs
conv3                (None, 128, 14, 14)     produces   25088 outputs
pool3                (None, 128, 7, 7)       produces    6272 outputs
hidden4              (None, 200)             produces     200 outputs
hidden5              (None, 200)             produces     200 outputs
output               (None, 30)              produces      30 outputs
```

Fig 5 : Layer wise input and output dimensions

To consume less computing resources, we reduce the number of nodes in hidden layers from 500 to 200(still more than our first CNN model).
This time we train the new structure for 50 epochs and we get results close to our last CNN model(with 500 hidden layer nodes) which ran for 100 epochs. Guided by this new advancement, we try again and this time we train the new structure for 100 epochs and to our excitement it outperformed the old model. Not only the accuracy got better but also the computation time went down to one third of what our initial model took. This shows how adjusting different parameters in separate layers of a CNN can help us achieve desired results together with consuming less resources. No doubt that with greater computation power, running our new model for more nodes in hidden layer for a larger number of epochs makes it more accurate. All thanks to our quest for a better network !

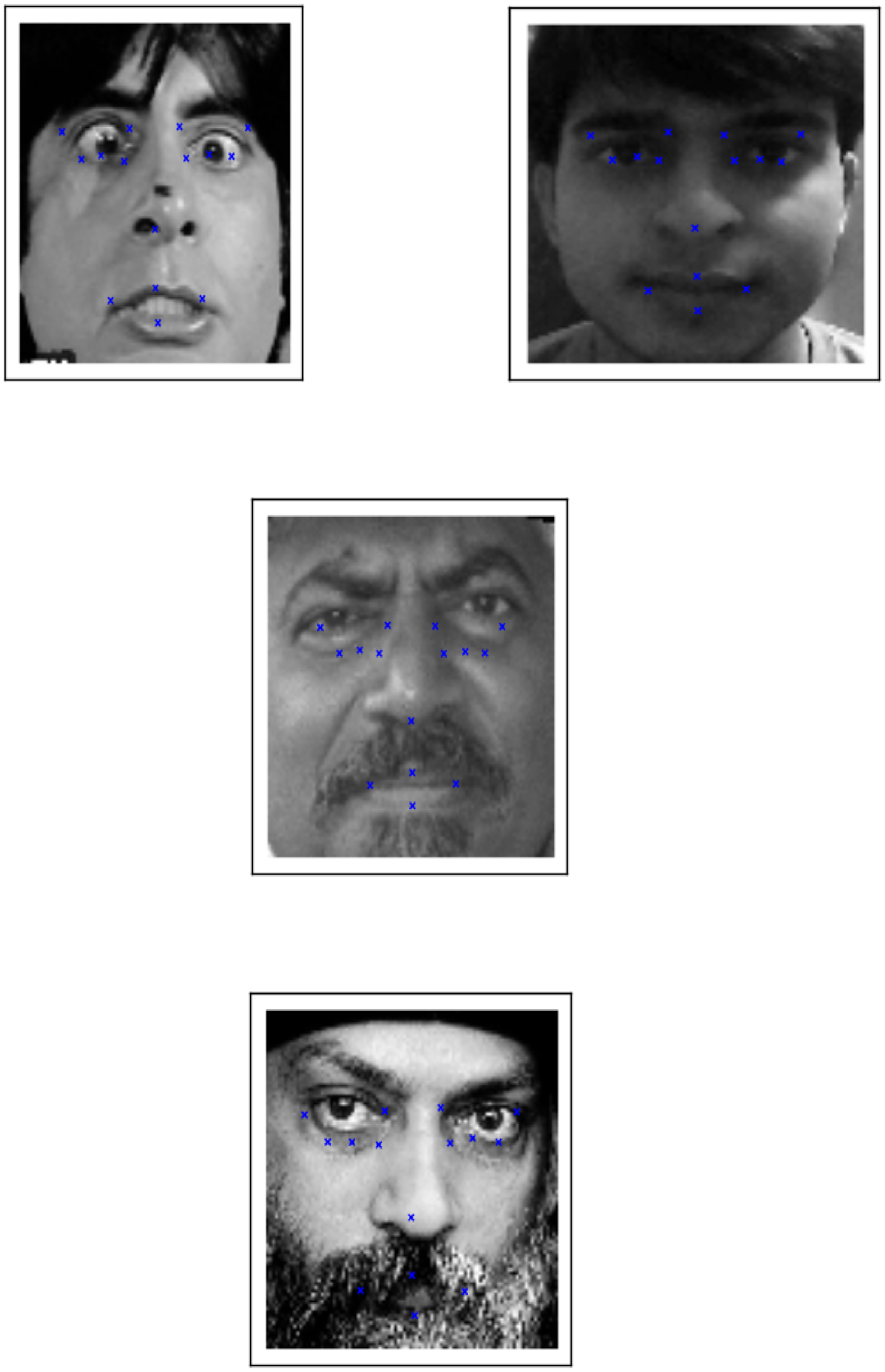
Results and Comparison

Earlier CNN model :

Training epochs	Validation loss	Hidden layer units	RMSE
100	0.004730	100	3.3014
100	0.004194	500	3.1085

Results with CNN after new improvement - changing the size of the filters in convolutional layers:

Training epochs	Validation loss	Hidden layer units	RMSE
50	0.004250	200	3.1292
100	0.004137	200	3.0873



Results with some Indian faces

Conclusion

We conclude that CNN, which is indeed a faster and more accurate method of deep learning, when applied to facial keypoint recognition gives results far too better than simple networks. Also that the approach is not affected by imaging factors like light intensity, camera angle, facial poses,etc. thus setting it different from many other methods.
Adjusting filter sizes, keeping larger filters for data input layers and decreasing the size in subsequent layers, produces better results. With further tuning of networks and large data size, the network is bound to improve.

Acknowledgements

We are very grateful to Prof. Amitabha Mukerjee for supporting and guiding us all the way for the challenge. All the understanding about neural networks and deep learning comes from his valuable lectures. Going over the research papers(below in references) made us clear about the wonderful applications of the technique. In the end Daniel Nouri’s blog was of great help for understanding how to tune a network.

Future work

There is a lot we can do further to decrease the RMSE :

1. We can simply try changing nodes in the hidden layer of the new improved structure to 500 or even more to reach an optimal value.
2. With better computing resources, we can train the structure for at least 1000 epochs and then the output RMSE will be definitely very good.
3. One main problem with our training data is that the data we have is of less complete samples(2140) than the input layer size (9216) , so we are losing some accuracy due to overfitting too. The best way to overcome that is to just flip the training data images after loading about the vertical axis thus generating more samples without blowing up memory.
4. From Ref[2] we know “For neurons in the convolutional layers,absolute value rectification after the hyperbolic tangent activation function can effectively improve the performance” only that it increases the computation time.
5. Adding some layers of pre-processing just before the two ending hidden layers can help reduce the complexity of the problem together with gaining accuracy.

References

1. Sun, Y., Wang, X., & Tang, X. (2013). Deep convolutional network cascade for facial point detection. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
2. Andrew L Maas, Awni Y Hannun, and Andrew Y Ng, "Rectifier nonlinearities improve neural network acoustic models," in ICML Workshop on Deep Learning for Audio, Speech, and Language Processing (WDLASL 2013), 2013.
3. Daniel Nouri's blog- (http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-detect-facial-keypoints-tutorial/)
4. Kaggle Facial keypoints detection challenge. (May 2013 – Dec 2015)
5. R. Herpers, M. Michaelis , K.-H. Lichtenauer , and G. Sommer - Automatic Face and Gesture Recognition, IEEE Computer Society, 1996.
6. Human Face Recognition Found In Neural Network Based On Monkey Brains - Feb 13, 2015(http://www.technologyreview.com/view/535176/human-face-recognition-fodund-in-neural-network-based-on-monkey-brains/)
7. http://stackoverflow.com/questions/26513236/how-do-we-get-define-filters-in-convolutional-neural-networks
8. http://www.researchgate.net/post/How_to_decide_the_number_of_hidden_layers_and_nodes_in_a_hidden_layer