Project Report: Facial expression Recognition

Introduction: An overview of the project

In this project, we have developed convolutional neural networks (CNN) for a facial expression recognition task. The goal is to classify each facial image into one of the seven facial emotion categories considered in this study. We trained CNN models with different depth network using gray-scale images from the Kaggle website. We developed our models in caffe and exploited Graphics Processing Unit (GPU) computation in order to expedite the training process To reduce the overfitting of the models, we utilized different techniques including dropout and batch. We applied cross validation to determine the optimal hyper-parameters and evaluated the performance of the developed models by looking at their training histories. We also present the visualization of some layers of a network to show what features of a face can be learned by CNN models.

As for the CNN models, we tried Lenet, alexnet, googlenet and resnet. Based on their different performance, we mainly used alexnet as our main network to do predictions.

Background

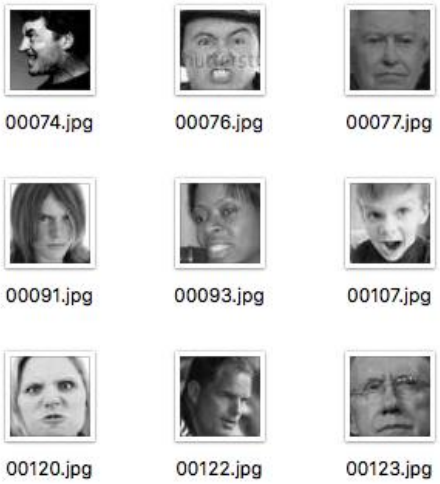
Humans interact with each other mainly through speech, but also through body gestures, to emphasize certain parts of their speech and to display emotions. One of the important ways humans display emotions is through facial expressions which are a very important part of communication. Though nothing is said verbally, there is much to be understood about the messages we send and receive through the use of nonverbal communication. Facial expressions convey nonverbal cues, and they play an important role in interpersonal relations. Automatic recognition of facial expressions can be an important component of natural human-machine interfaces; it may also be used in behavioral science and in clinical practice. Although humans recognize facial expressions virtually without effort or delay, reliable expression recognition by machine is still a challenge. There have been several advances in the past few years in terms of face detection, feature extraction mechanisms and the techniques used for expression classification, but development of an automated system that accomplishes this task is difficult. In this project, we present an approach based on Convolutional Neural Networks (CNN) for facial expression recognition. The input into our system is an

image; then, we use CNN to predict the facial expression label which should be one these labels: anger, happiness, fear, sadness, disgust and neutral.

In this project, we used fer2013 as our dataset to train the model and also test with it.

Description of Data : FER2013

The dataset is FER2013, obtained from Kaggle. The data consists of 48x48 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 task is to categorize each face 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=Normal).



The training set consists of 28,709 examples. The public test set used for the leaderboard consists of 3,589 examples. The final test set, which which was used to determine the performance of the predicting, consists of another 3,589 examples. This dataset was prepared by Pierre-Luc Carrier and Aaron Courville.

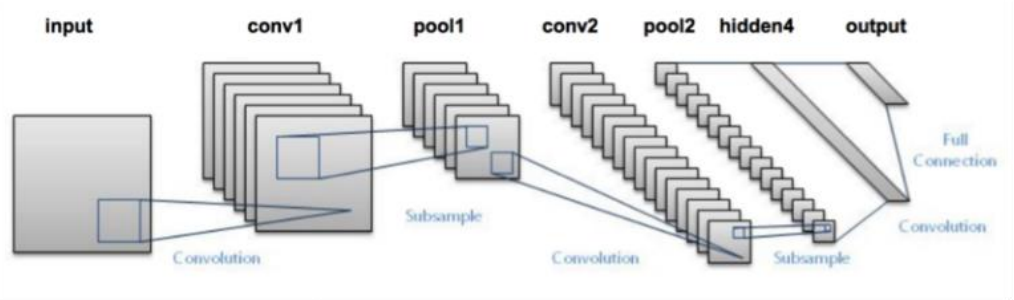
The reason why we choose this dataset:

1. The size of this dataset is moderate;
2. Human faces take a large place of the pictures, so corp may not be necessary

Description of the deep learning network and training algorithm

LeNet

The LeNet architecture was first introduced by LeCun et al. in their 1998 paper, Gradient-Based Learning Applied to Document Recognition. The LeNet architecture is straightforward and small, making it perfect to give an overall evaluation of the dataset.



The network of LeNet is quite simple: two convolution layers and one fully connected layer.

AlexNet

AlexNet. The first work that popularized Convolutional Networks in Computer Vision was the AlexNet, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. The AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error). The Network had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other (previously it was common to only have a single CONV layer always immediately followed by a POOL layer).

Figure 1.The prototype of AlexNet

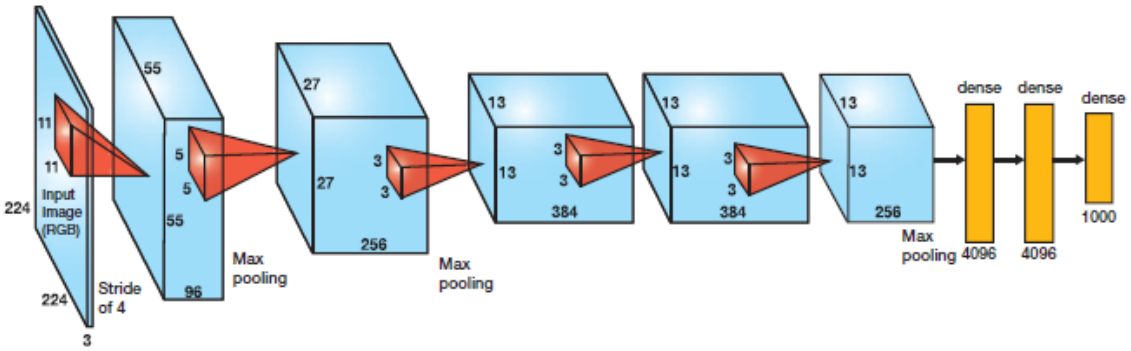
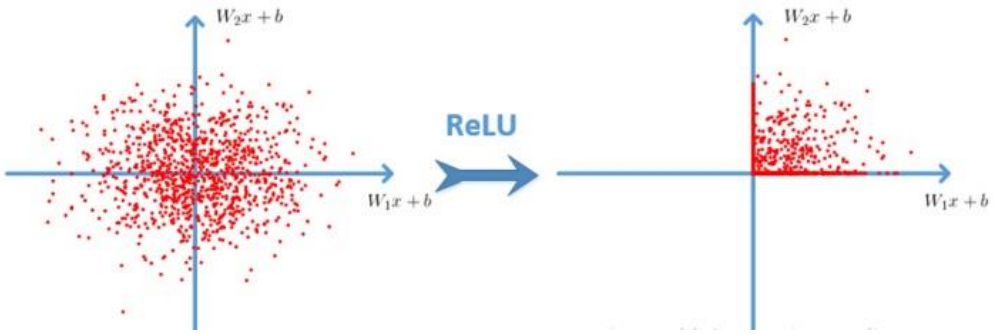


Table 1.The structure of AlexNet used in this project

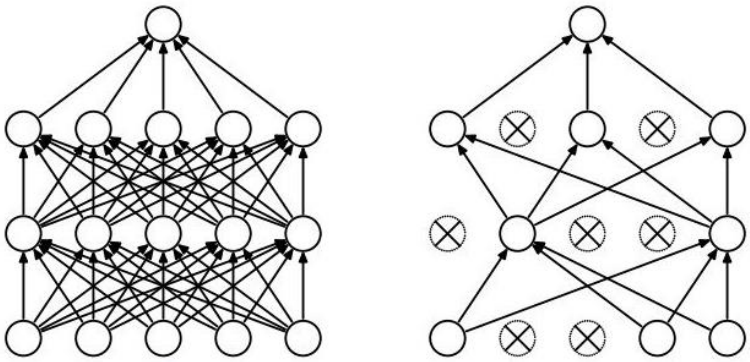
Type	Conv+ max+ norm	Conv+ max+ norm	conv	conv	Conv+ max	full	full	full
Channels	96	256	384	384	256	4096	1000	7
Filter Size	11*11	5*5	3*3	3*3	3*3	-	-	-

Convolution Stride	4*4	1*1	1*1	1*1	1*1	-	-	-
Pooling Size	3*3	3*3	-	-	3*3	-	-	-
Pooling Stride	2*2	2*2	-	-	2*2	-	-	-
Padding Size	2*2	1*1	1*1	1*1	1*1	-	-	-

As above table shows, AlexNet has 8 layers, 5 convolution layers and 3 fully connected layers. AlexNet, uses ReLu(Rectified Linear Unit) for the non-linear part, instead of a Tanh or Sigmoid function which was the earlier standard for traditional neural. it trains much faster than the latter because the derivative of sigmoid becomes very small in the saturating region and therefore the updates to the weights almost vanish.



Another problem that this architecture solved was reducing the over-fitting by using a Dropout layer after every FC layer. Dropout layer has a probability,(p), associated with it and is applied at every neuron of the response map separately. It randomly switches off the activation with the probability **p.**



(a) Standard Neural Net (b) After applying dropout.

Alexnet also adopts a Local Response Normalization (LRN) after each convolution layers. LRN can be justified as an approximate whitening based on the assumption of a high degree of correlation between neighbouring pixels. The author claims that by using LRN, top-1 and top-5 error rates were reduced by 1.4% and 1.2% respectively for ImageNet classification.

Experimental setup

Data prepare: convert to lmdb

The dataset provided by Kaggle is a csv file. It contains three columns, the first column indicate the categories of the picture, the second is the pixels of the according picture and the third column has labels show that if this picture is a training set, validation set or test set.

1. Convert the pixel column to .JPG format pictures and store them to train, val and test folders.
2. Generate index .txt files of each dataset. Showing the relative directory and name of the pictures and its categories. This step is to prepare lmdb data. Here we create a bash script *create_aux.sh* to generate the auxiliary .txt data.

```
#!/bin/bash

ls train/0 | sed "s:^:0/:" | sed "s:$: 0:" >> train.txt
ls train/1 | sed "s:^:1/:" | sed "s:$: 1:" >> train.txt
ls train/2 | sed "s:^:2/:" | sed "s:$: 2:" >> train.txt
ls train/3 | sed "s:^:3/:" | sed "s:$: 3:" >> train.txt
```

So we could simply run the script and get the following outcome.

```
0/00111.jpg 0
0/00112.jpg 0
0/00121.jpg 0
0/00122.jpg 0
0/00131.jpg 0
```

3. Then we copy file *create_imagenet.sh* from ~/caffe/build/tools and modify the paths to our datasets path and specify the lmdb file names and destinate

locations. Then we could generate lmdb format dataset. We also resize the picture size to 227 while we use ResNet, because a 48*48 matrix is not enough for such a large network.

```
EXAMPLE=/home/jingsi/ml2_final_project/new_fer2013/datasets
DATA=/home/jingsi/ml2_final_project/new_fer2013/datasets
TOOLS=/home/jingsi/caffe/build/tools

TRAIN_DATA_ROOT=/home/jingsi/ml2_final_project/new_fer2013/datasets/train/
VAL_DATA_ROOT=/home/jingsi/ml2_final_project/new_fer2013/datasets/val/
```

```
RESIZE=false
if $RESIZE; then
  RESIZE_HEIGHT=48
  RESIZE_WIDTH=48
else
  RESIZE_HEIGHT=0
  RESIZE_WIDTH=0
```

```
Creating train lmdb...
I0424 23:52:08.379891 30003 convert_imageset.cpp:86] Shuffling data
I0424 23:52:09.273877 30003 convert_imageset.cpp:89] A total of 57418 images.
I0424 23:52:09.277016 30003 db_lmdb.cpp:35] Opened lmdb /home/jingsi/ml2_final_p
roject/new_fer2013/datasets/fer2013_train_lmdb_resize
I0424 23:52:28.633476 30003 convert_imageset.cpp:147] Processed 1000 files.
I0424 23:52:46.115595 30003 convert_imageset.cpp:147] Processed 2000 files.
I0424 23:53:02.963749 30003 convert_imageset.cpp:147] Processed 3000 files.
```

4. After we have our datasets, we still need a mean value for the network when compute pictures. Again, we modify *make_imagenet_mean.sh* from ~/caffe/build/tools to calculate the mean value of all the pictures in the training set.

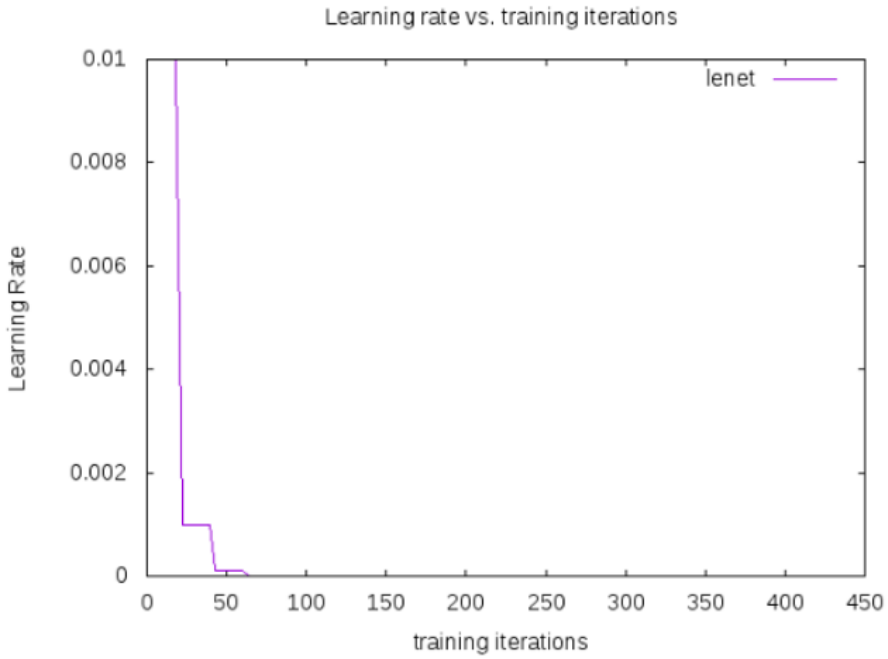
```
/data_prepare$ ls
create_aux.sh               make_imagenet_mean.sh
create_imagenet.sh          Resnet_50_pretrain.prototxt
fer2013_mean.binaryproto    Resnet_pretrain_solver.prototxt
lenet_deploy.prototxt       test.txt
lenet_solver.prototxt       train.txt
lenet_train_test.prototxt   val.txt
```

These are all the files we have after data preparation.

Training

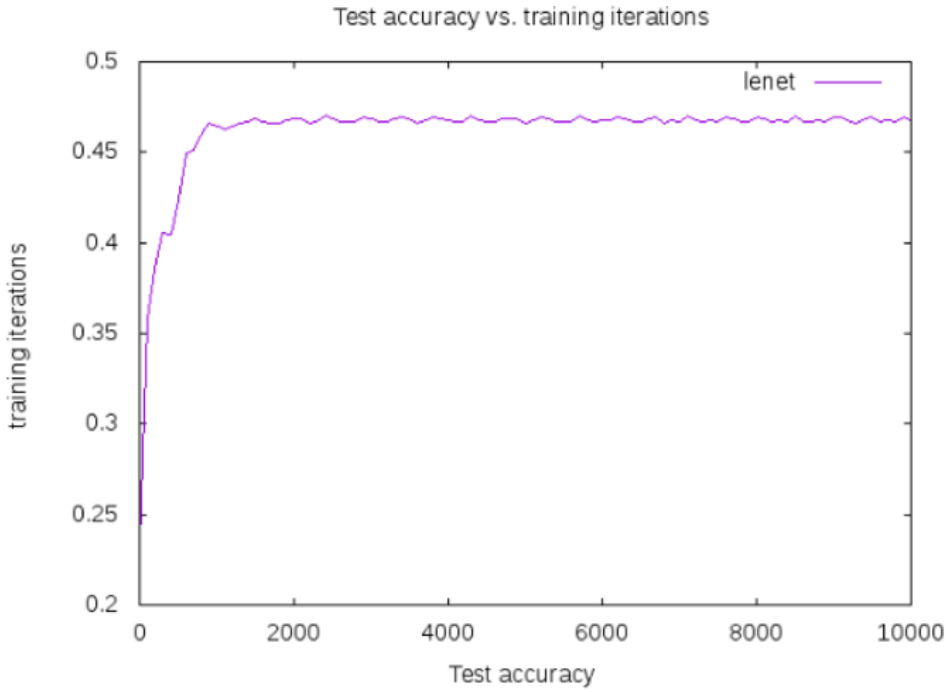
LeNet

After we finish the data, we start a test training with LeNet. Because LeNet is a simple and classic convolutional neural network. It will not take too much time running and will provide a baseline of our later work.
We first use the solver file we used in exam2 and use max iteration equals 10000. As we can see, the learning rate drops two fast and we can hardly know if the network still learning something after 100 iteration.

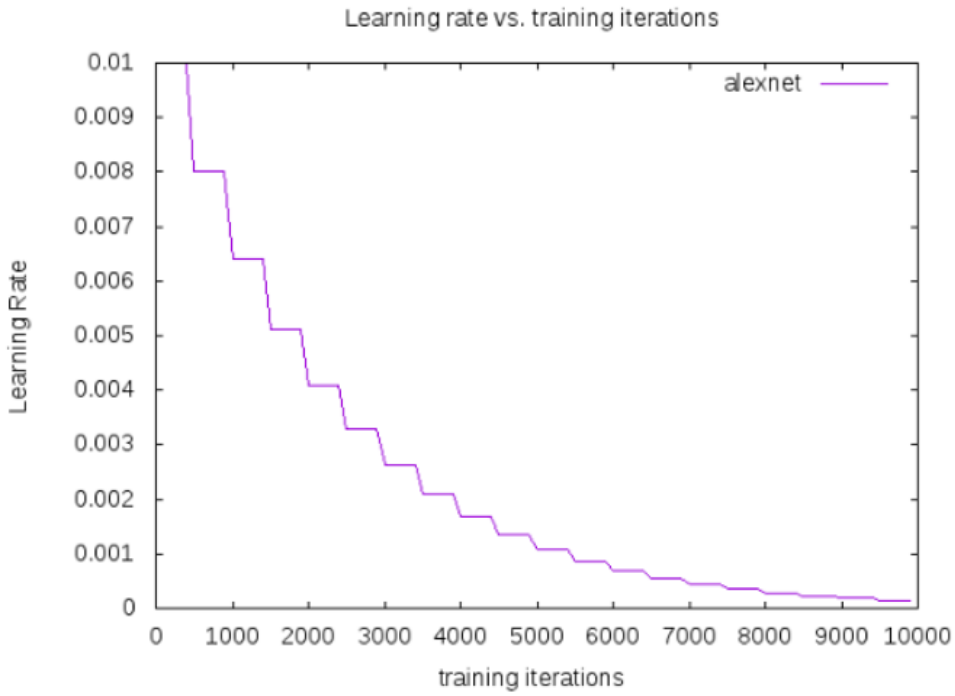


This bad learning rate also leads to a very bad result. The training and validation loss almost stay the same. The model almost does not learn anything after about 1000 iterations.

The same with the test accuracy. Almost after 1000 iteration, the model start to converge, which makes the later running meaningless. However, the convergence also related to batch size. We will talk about this later.



Since the learning rate is so inappropriate, we change decay method to “step” instead of “inv”. We set the gamma to 0.9 and step size to 500. And get this much better graph.



Still, this learning rate is not very good. It seems like it to decrease too slow. But this gives us a basic idea of where to modify it.

As we can see, the training loss has a major decrease. But the validation set does not perform well. Seems like we have a overfit here. It is better to add a dropout layer. We have this in our later work of AlexNet.

Compare to the previous one, this is gives a better result. It seems like LeNet has reached its limits. By now, we have some overall idea of our project.

Alexnet

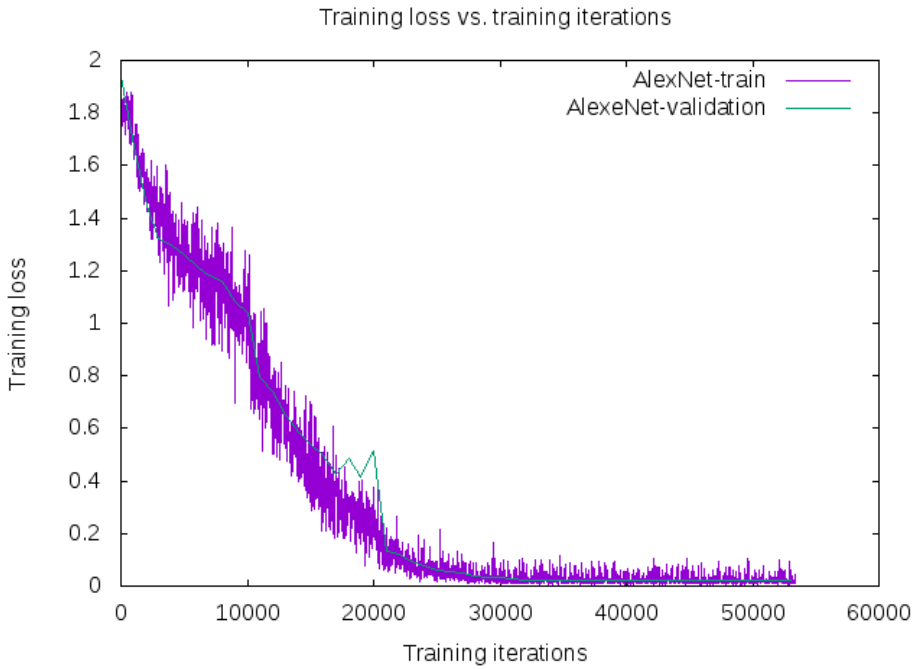
In this project ,we trained 3 AlexNets with training set of **fer 2013** data set. We changed some parameters in each network for exploring a better parameter selection for this data set.

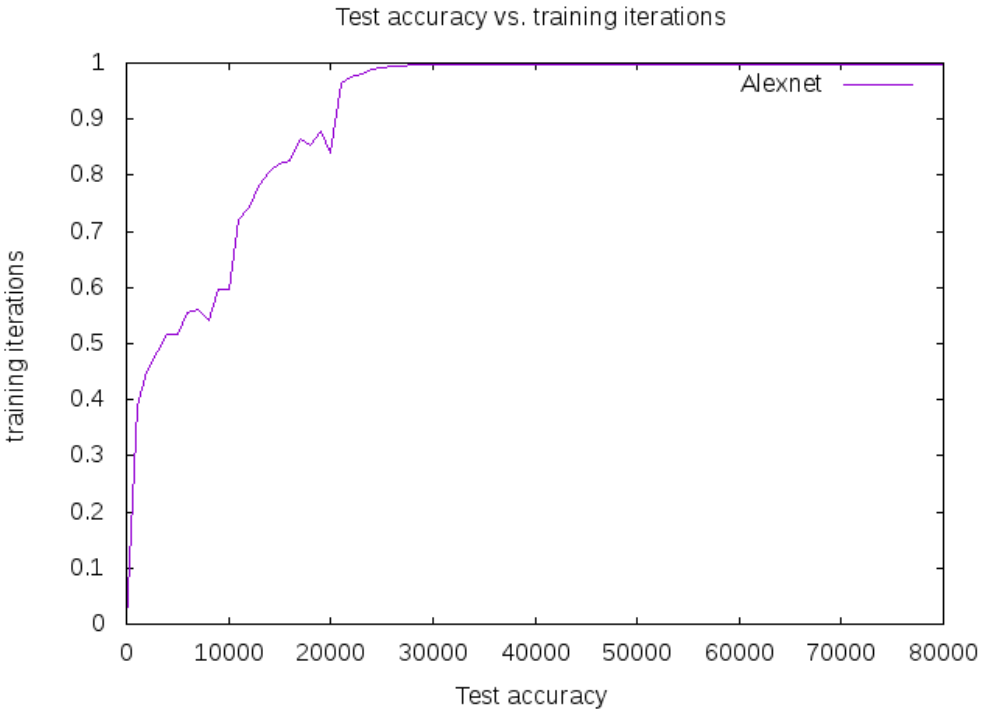
Table 2. Parameter of three AlexNets

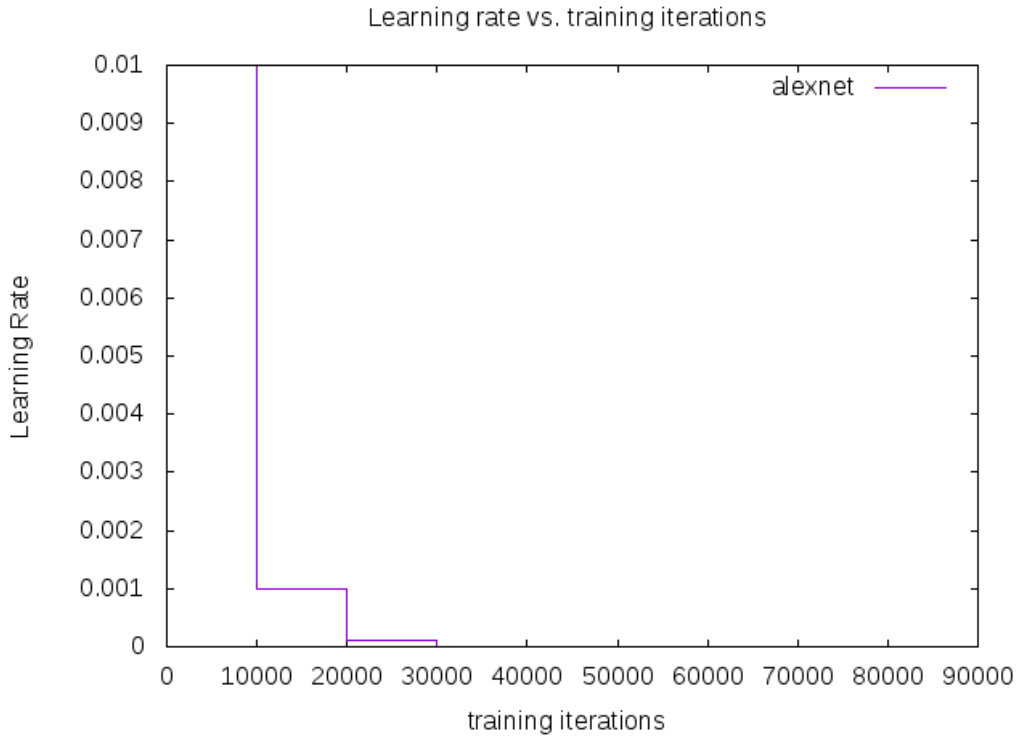
	Iterations	**Training batch size**	**Gamma (lr_policy:step)**
AlexNet 1	50000	100	0.1
AlexNet 2	50000	256	0.1
AlexNet 3	50000	100	0.95

After training each network for 50000 iterations, we used the log file of them printing the 3 plots(Training loss vs. training iterations, Test accuracy vs. training iterations, Learning rate vs. training iterations) for each network which are showed following.

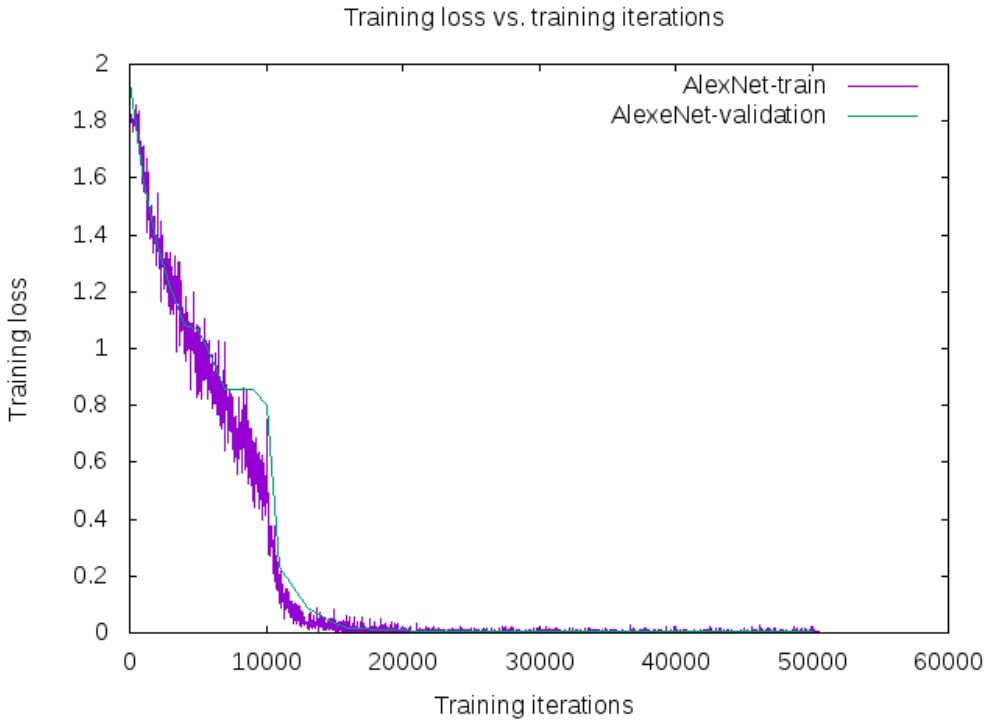
For AlexNet 1:

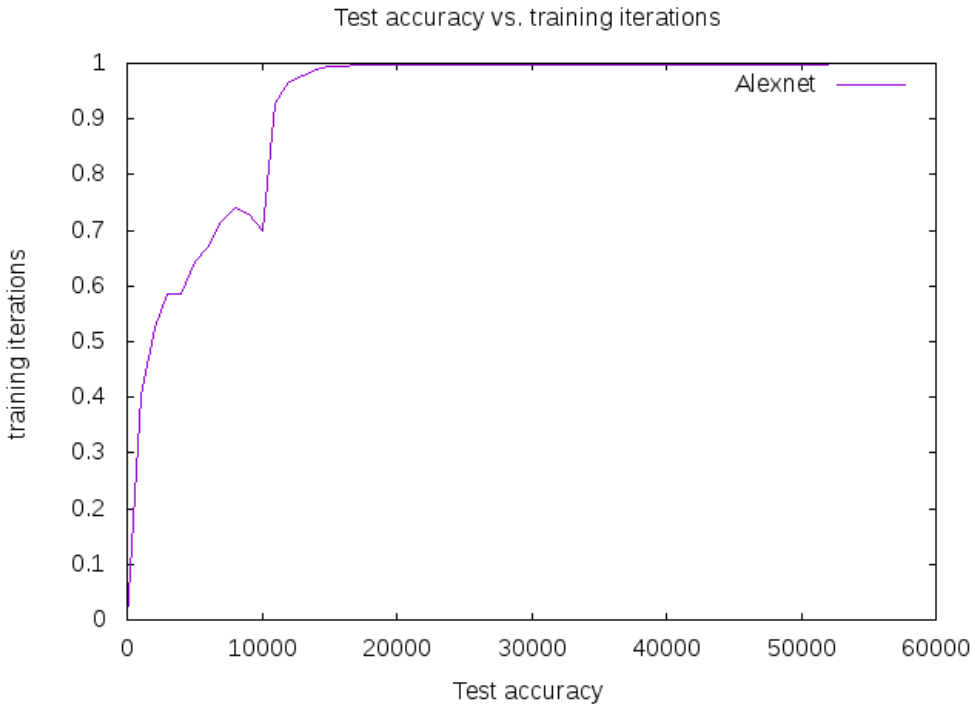

Training loss vs. training iterations
AlexNet-train
AlexeNet-validation
Training loss
Training iterations
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
0
10000
20000
30000
40000
50000
60000

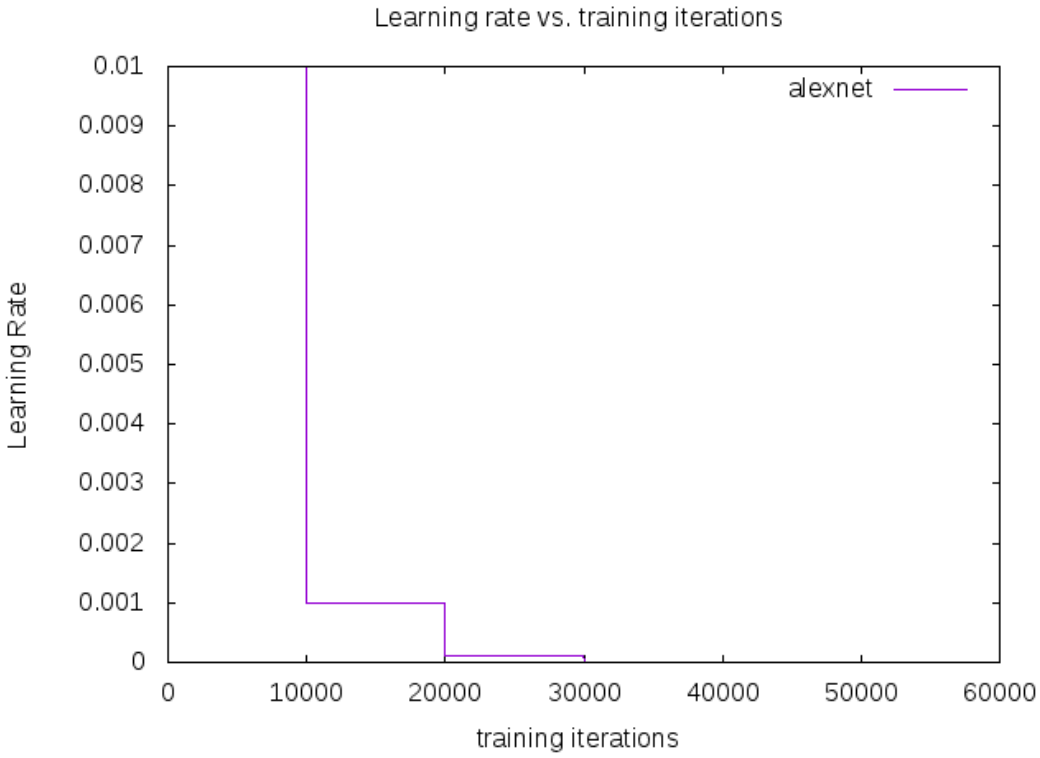

Test accuracy vs. training iterations
Alexnet
training iterations
Test accuracy
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
10000
20000
30000
40000
50000
60000
70000
80000



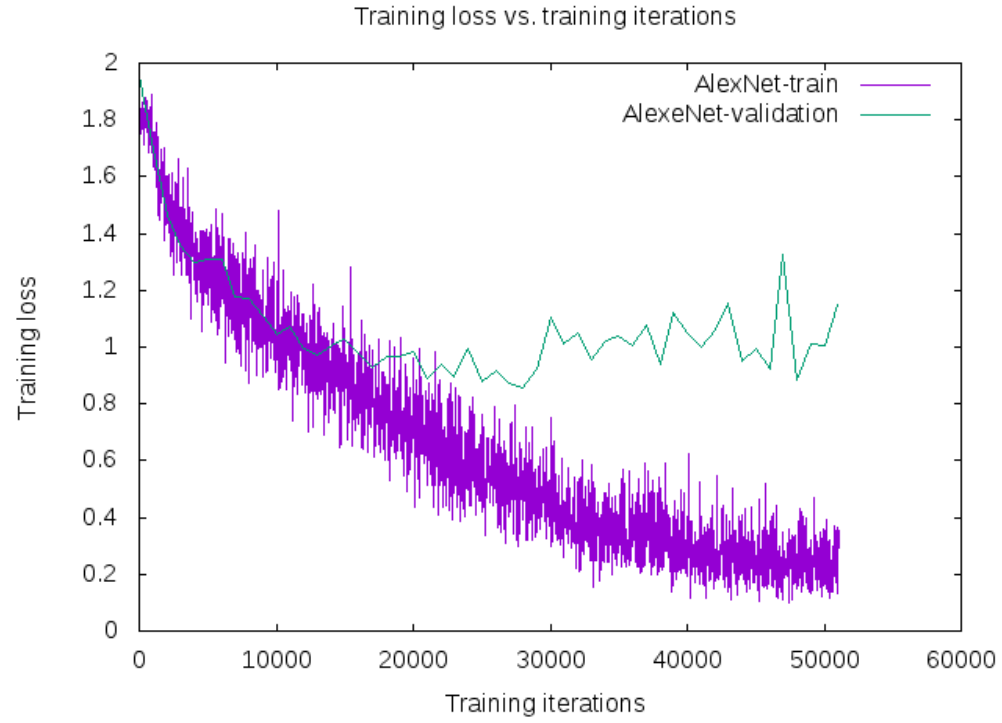
For AlexNet 2:

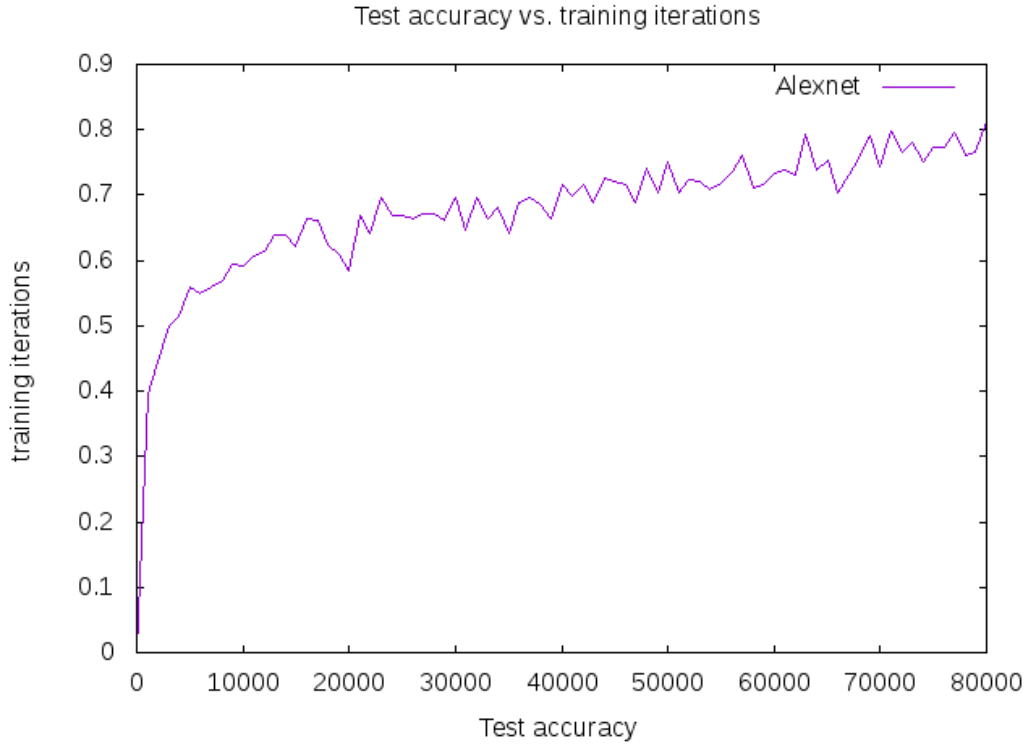


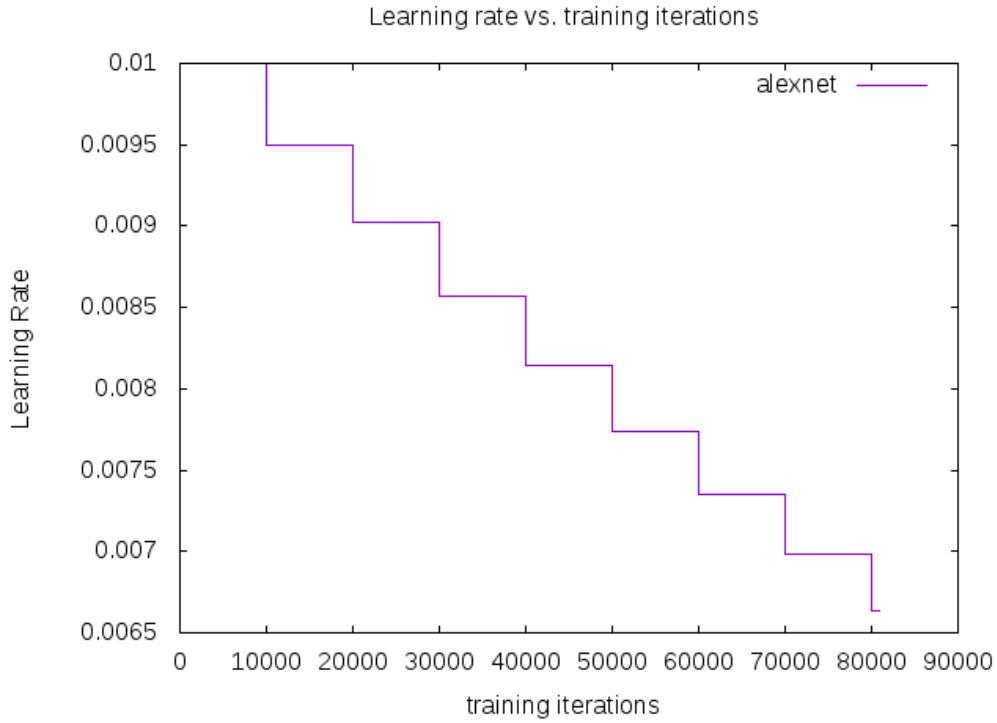




For AlexNet 3:


Training loss vs. training iterations
AlexNet-train
AlexeNet-validation
Training loss
Training iterations
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
0
10000
20000
30000
40000
50000
60000


Test accuracy vs. training iterations
Alexnet
training iterations
Test accuracy
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
10000
20000
30000
40000
50000
60000
70000
80000



Above plots shows some interesting results for us. The training batch size of **AlexNet 2** is bigger than **AlexNet 1**, which reflected on the plot is that **AlexNet 2** converges faster to a steady status than **AlexNet 1. To a certain extent, the bigger Batch Size, the more accurate it is in the direction of descent, resulting in less training turbulence.**

For **AlexNet1** and **AlexNet 3**, we used a different learning rate decay speed. We used step as the learning rate policy, which learning rate can be calculated by **base_lr * gamma ^ (floor(iter / stepsize))**. As the plot shows, the learning rate of AlexNet 3 decays slower than AlexNet 1， which leads to an overfitting phenomenon on AlexNet 3 because the learning rate of Alex Net 3 is too large when the network goes to more than 10000 iterations.

ResNet-50:

This is a modified version of ResNet. Their model is claimed by a few person of Media and Cognition of Department of EE., Tsinghua University to have 97.7% accuracy on open datasets including “fer2013”, the one we use here. But we underestimate the complexity of this model. After over 50 hours running, we still far from the end. The rough estimate running time on a single GPU on our GCP instance is over 210 hours. So we plan to run on multiple GPUs. Till right now, at about 1/7 of the progress, the model gives an accuracy on validation dataset of 55%.

Deploy prototxt:
The deploy file is a simple modify with the train prototxt. We use it to apply our trained model on test dataset.

Analysis and visualization

AlexNet

The best performance we got with Alexnet:

Accuracy: 54.8620785734%

Misclassification Rate: 45.1379214266%

Based on the output (response) of the network, we plotted confusion matrices and ROC curves (Receiver Operating Characteristic) and calculated AUC (area under ROC curve) to intuitively visualize our network predictions(outputs) and accuracy for each class. (we have seven classes of emotion)

- **Confusion matrix**

[[203 6 67 43 75 7 66]

 [16 20 4 5 6 0 5]

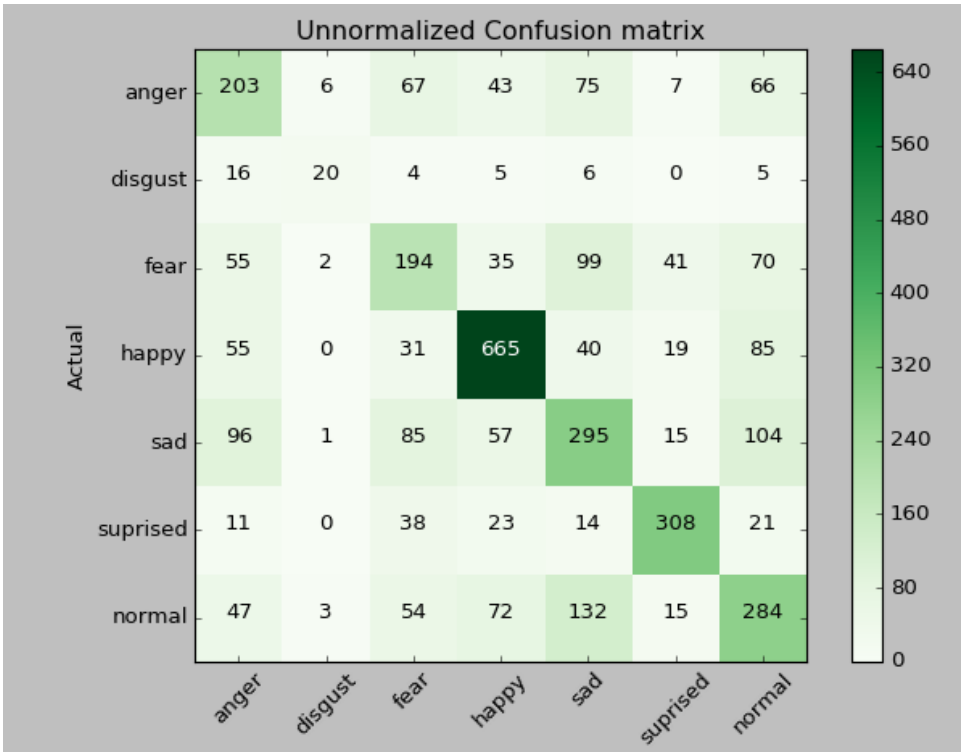
 [55 2 194 35 99 41 70]

 [55 0 31 665 40 19 85]

 [96 1 85 57 295 15 104]

 [11 0 38 23 14 308 21]

 [47 3 54 72 132 15 284]

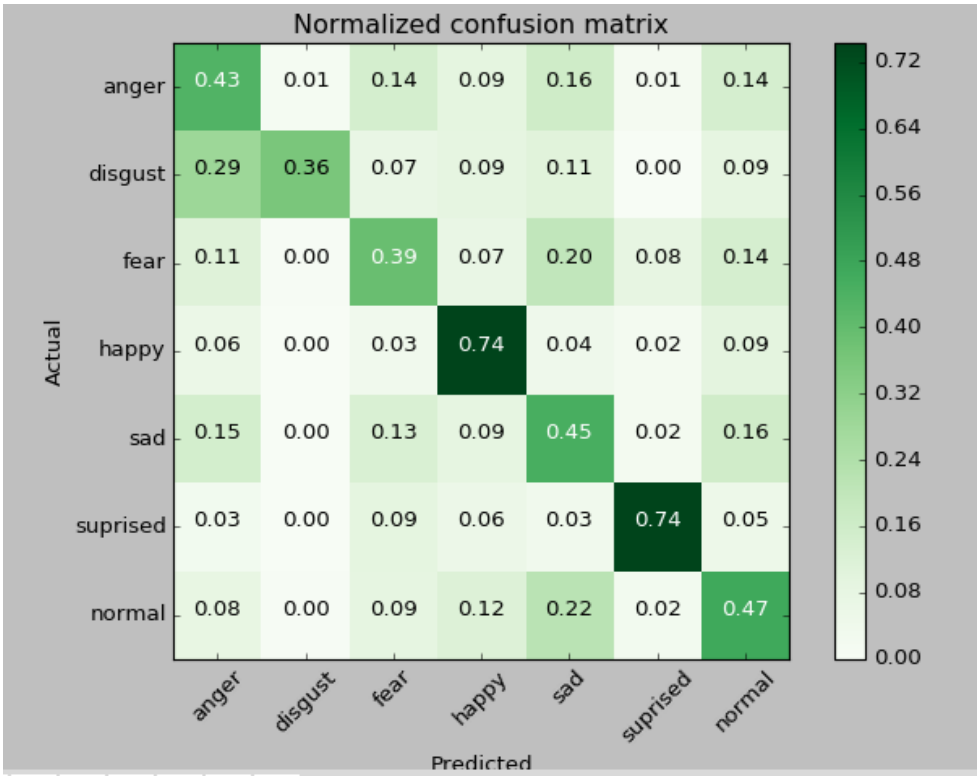


- **Expected down the side**: Each column of the matrix corresponds to a predicted class.
- **Predicted across the top**: Each row of the matrix corresponds to an actual class.

The counts of correct and incorrect classification are then filled into the table.
The total number of correct predictions for a class go into the expected row for that class value and the predicted column for that class value.
In the same way, the total number of incorrect predictions for a class go into the expected row for that class value and the predicted column for that class value.

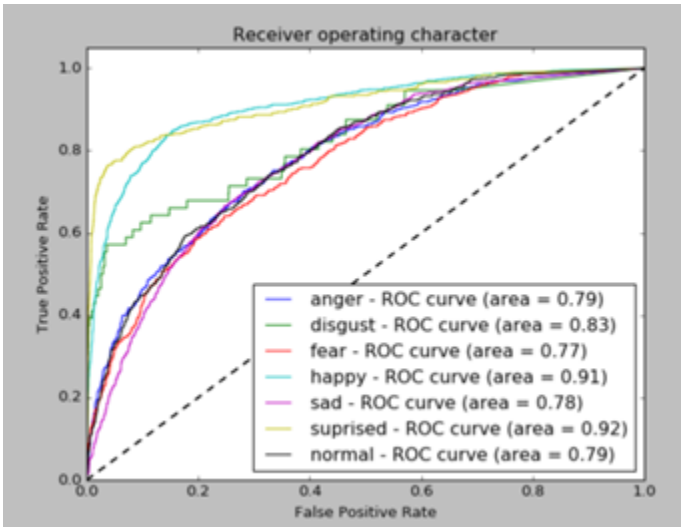
A confusion matrix is a summary of prediction results on a classification problem.It can give us a better idea of what our classification model is getting right and what types of errors it is making, which is very intuitive.

The number of correct and incorrect predictions are summarized with count values and broken down by each class (we have seven classes). This is the key to the confusion matrix.
In this way, it gives us insight not only into the errors being made by my classifier but more importantly the types of errors that are being made. It is this breakdown that overcomes the limitation of using classification accuracy alone in our predicting.

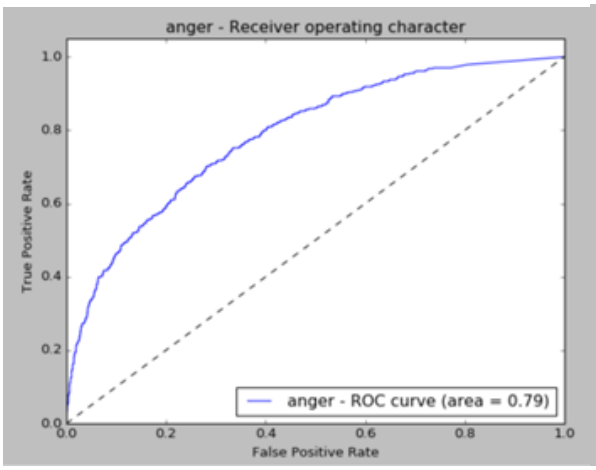


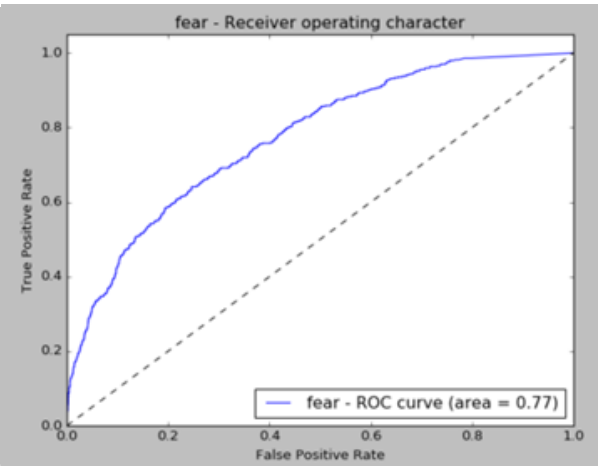
Here is a normalized version of the confusion matrix.
From this plot, we can tell more detailed information about our prediction. Each element on the diagonal indicates the accuracy of each class specifically. And the sum of each row is equal to one, which represents all of the test images for that class in the test set.

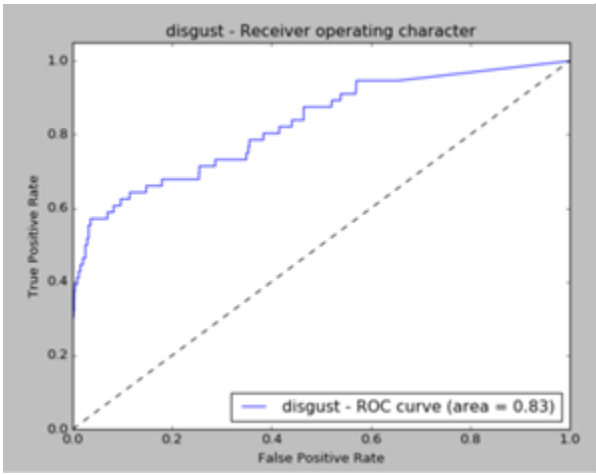
- **ROC Curve & AUC Value**

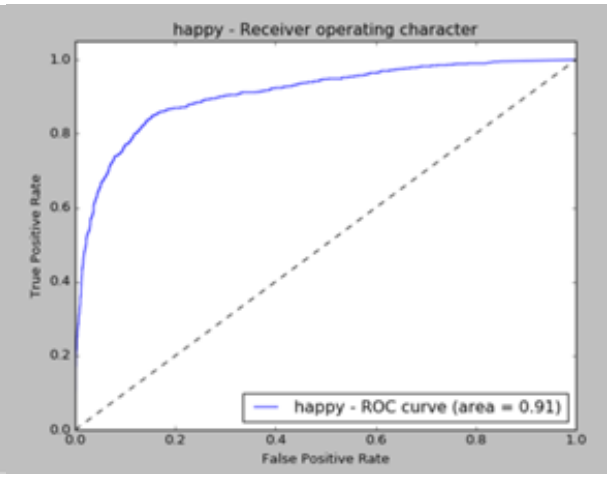


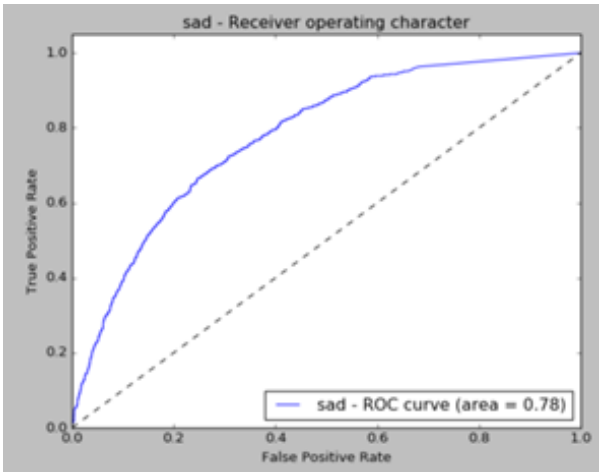
When we select a fixed false positive rate, if the true positive rate is higher, the performance (accuracy) is better.

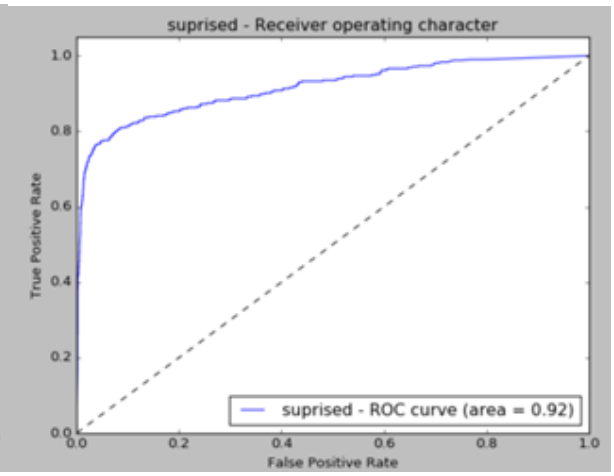

anger - Receiver operating character
True Positive Rate
False Positive Rate
anger - ROC curve (area = 0.79)

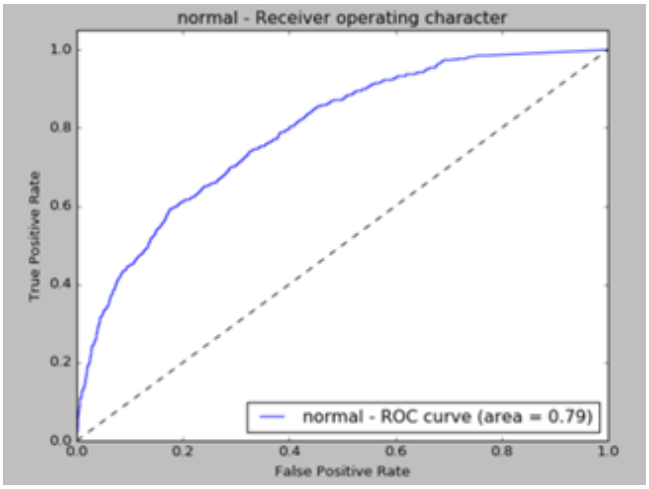

fear - Receiver operating character
True Positive Rate
False Positive Rate
fear - ROC curve (area = 0.77)


disgust - Receiver operating character
True Positive Rate
False Positive Rate
disgust - ROC curve (area = 0.83)


happy - Receiver operating character
True Positive Rate
False Positive Rate
happy - ROC curve (area = 0.91)


sad - Receiver operating character
True Positive Rate
False Positive Rate
sad - ROC curve (area = 0.78)


suprised - Receiver operating character
True Positive Rate
False Positive Rate
suprised - ROC curve (area = 0.92)



An ROC curve is the most commonly used way to visualize the performance of a binary classifier (a classifier with two possible output classes), so for our network, we have seven different Roc curves corresponding to seven different classes in our target. And AUC is (arguably) the best way to summarize Roc curve performance in a single number. Generally speaking (based on the general experience, the performance of the prediction is good if the AUC is higher than 0.75.

ROC curve is a plot of the true positive rate against the false positive rate for the different possible thresholds (cut points) of a diagnostic test.

And an ROC curve usually demonstrates several things:

1. It shows the tradeoff between true positive rate (sensitivity) and false positive rate(specificity) (any increase in sensitivity will be accompanied by a decrease in specificity).
2. The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test.
3. The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.
4. The area under the curve is a measure of test accuracy, also called AUC(area under ROC curve).

As for the AUC (area under ROC curve), which is used to measure the accuracy. An area of 1 represents a perfect test; an area of .5 represents a worthless test. A rough guide for classifying the accuracy of a diagnostic test is the traditional academic point system:

- .90-1 = excellent (A)
- .80-.90 = good (B)
- .70-.80 = fair (C)
- .60-.70 = poor (D)
- .50-.60 = fail (F)

Summary and conclusions

Generally speaking, the performance we got with alexnet is not bad, kind of good. The accuracy is high enough for us. We can conclude that deep neural networks can really do a good job on classifications and predictions. In the process of doing this project, which we enjoyed very much, we got very well familiar with caffe framework and alexnet network structure. For the future, we may want to try more complicated convolution networks like googlenet, resnet and spend much longer time on training the network to get a much higher accuracy. In addition, we would like to try some other frameworks like pytorch that we use in the exam1 and compare the performance with caffe.

References

1. Caffe models of Berkeley Vision and Learning Center
https://github.com/BVLC/caffe/tree/master/models
2. Challenges in Representation Learning: Facial Expression Recognition Challenge
https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data
3. ResNet, AlexNet, VGGNet, Inception: Understanding various architectures of Convolutional Networks
http://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/
4. http://www.cnblogs.com/hust-yingjie/p/6528320.html
5. Importance of local response normalization in CNN
https://stats.stackexchange.com/questions/145768/importance-of-local-response-normalization-in-cnn?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa
6. https://zhuanlan.zhihu.com/p/21562756
7. https://blog.csdn.net/qq_20259459/article/details/70316511
8. Facial Expression Recognition from World Wild Web https://arxiv.org/pdf/1605.03639.pdf
9. Going Deeper in Facial Expression Recognition using Deep Neural Networks
https://arxiv.org/pdf/1511.04110.pdf
10. Facial Expression Recognition based on caffe
https://blog.csdn.net/pangyunsheng/article/details/79434301
11. A step by step guide to Caffe
http://shengshuyang.github.io/A-step-by-step-guide-to-Caffe.html
12. Media and Cognition course project, facial expression recognition using ResNet-50
https://github.com/ybch14/Facial-Expression-Recognition-ResNet
14. Caffe Logging and Loss Plotting
https://github.com/BVLC/caffe/blob/master/tools/extra/plot_log.gnuplot.example
15. confusion matrix
https://gist.github.com/Coderx7/830ada70049ebbe7d7dc75101645b2f9#file-caffe_convnet_confuxionmatrix-py
16. Using a Trained Network: Deploy
https://github.com/BVLC/caffe/wiki/Using-a-Trained-Network:-Deploy

Appendix (documented computer listings).

Code for this project can be found at the following github repo

For Training AlexNet
train_AlexNet.py
AlexNet_deploy.prototxt
Fer2013_mean.binaryproto

For AlexNet 1
AlexNet.prototxt
AlexNet_solver.prototxt
Alex_base_Confusion_matrix.sh
Alex_base_ROC.sh
pretrain_Alex.sh
AlexNet1_output_code

For AlexNet 2
AlexNet_256_size.prototxt
AlexNet_256_size_solver.prototxt
Alex_256_Confusion_matrix.sh
Alex_256_ROC.sh
pretrain_Alex_256_size.sh
AlexNet2_output_code

For AlexNet 3
AlexNet_gamma0.95.prototxt
AlexNet_gamma0.95_solver.prototxt
Alex_gamma0.95_Confusion_matrix.sh
Alex_gamma0.95_ROC.sh
pretrain_Alex_gamma0.95.sh
AlexNet3_output_code

For generating Confusion Matrix and ROC curve
ROC.py
confusion_matrix.py