

Machine Learning II Midterm Project

Classify Good or Bad Signal

Yue Zhou, Pengbo Cao

October 29th, 2017

Background

In embryonic development, it is important to spatiotemporally characterize molecular expression in order to have a better understanding of the process of cells becoming different and the cause of distinct tissues and organs. A single-cell technology, capillary electrophoresis(CE) and mass spectrometry(MS), were combined to reveal the abundance of each small molecule within a single cell with high resolution. While CE was used to separate different molecules in a tiny capillary based on their electrophoretic mobility with the use of an applied voltage, MS ionized chemical species and sorts the ions based on their mass-to-charge ratio.

The MS data from experiment is a 2D intensity array with one dimension standing for m/z while the other for RT. After some traditional signal processing approaches, we detect the potential targets with a low cutoff, so that we decrease the number of targets dramatically in order to make it feasible to look into the 2D details of each potential signals, to classify them into good or bad signals based on their signal shape. Since the diffusion property in capillary and the experimental error in the m/z evaluation in MS, the real signal shape is a smooth Gaussian curve in both dimension, thus forming a fine 2D Gaussian surface.

Data Description

The data consist of 4546 labeled sample grey scaled images, with only one channel instead of three channels. There are four set of data, each row of the dataset representing one image. One image size is 60×12 . *Imgs-train.txt* is the data representing the raw data, and the *bbs-train.txt* file is a bounding box for the highly likely region (with higher intensity), which contains most of the information provided by the original data. The image size for the bounding box is 40×20 using interpolation. *Label-train.txt* file which contains the label for each sample, i.e., whether the signal is bad or good. The *list-train.txt* file list some features about each sample. The four txt files are with the same order.

Models and Techniques

Our goal is to classify signals into two groups, either good or bad.

Following are the models that we have tried:

1. Logistic Regression
2. Multi fully-connected neurons with hidden layers
3. Convolutional Neural Network

Binary Logistic Regression

Binary logistic regression is classic method to estimate the probability of a binary response based on one or more predictors.

Multi Fully-connected Neurons with Hidden Layers

We started with 1-layer neural network along with 2 output neurons since we want to classify digits into 2 classes (good or bad), softmax as activation function, and cross-entropy as loss function. The accuracy is 0.68. To improve the recognition accuracy, we add more layers. We keep softmax as the activation function on the last layer since that is what works best for classification. On intermediate layers, we will use the most classic activation function: the sigmoid. The accuracy is 0.79633.

Convolutional Neural Network

Cross validation We continued with convolutional neural network with a five-layer. First three layers are convolutional layers, and last two layers are fully connected layers.

Following are the results for logistic regression, five-layer neural network, convolutional neural network, fig 1- 3, respectively.

Optimizer	Test Accuracy	AUC(Area under ROC Curve)
Logistics Regression	0.822018	0.87
5 layer NN	0.895413	0.96
5 layer CNN	0.922936	0.97

Table.1 model comparison

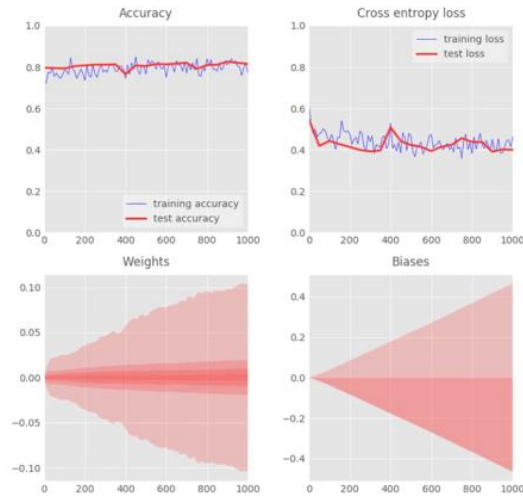


Figure.1 Logistics Regression analysis

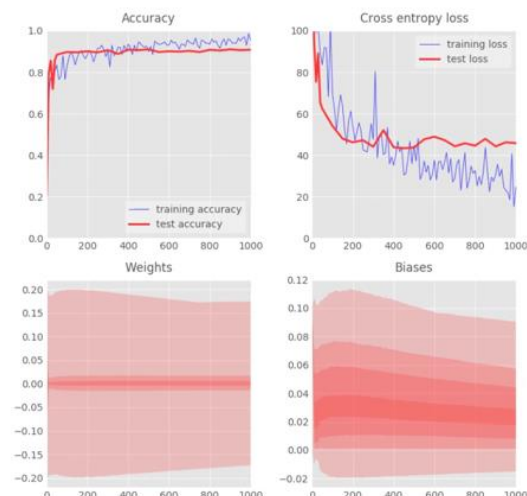


Figure.2 Five-layer NN

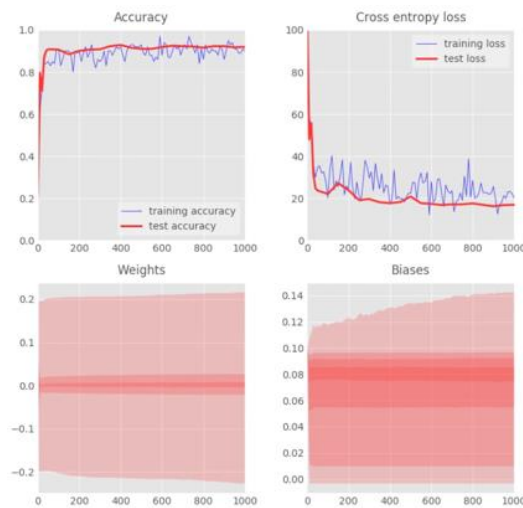


Figure.3 Five-layer CNN

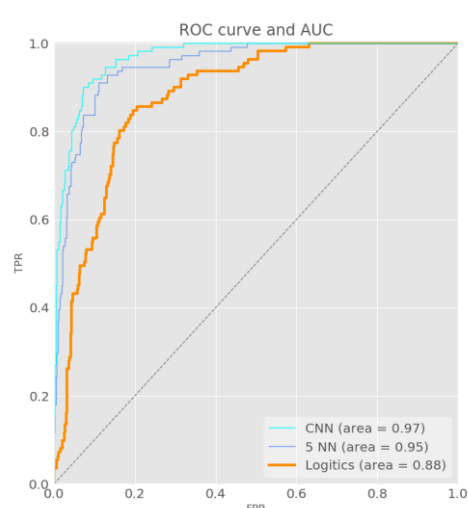


Figure.4 Different model ROC

After comparing these three models, we find convolutional neural network obtain higher accuracy and more stable than the other two. Therefore, we make several improvements based on convolutional neural network, as described below.

A few techniques we used:

1. Activation function: ReLU instead of Sigmoid
2. Changing learning rate: start fast and decay the learning rate exponentially
3. Tried different optimizer
4. Used random drop out in order not to over fit the data
5. Data preprocessing: Normalization
6. Data preprocessing: Bounding
7. CPU and GPU
8. Receiver operating characteristic(ROC)

ReLU Activation Function

The sigmoid activation function is actually quite problematic in deep networks. It squashes all values between 0 and 1 and when do so repeatedly, neuron outputs and their gradients can vanish entirely. Thus we use ReLU(Rectified Linear Unit) instead. The accuracy is still 0.79633.

Decay Learning Rate

By looking at the graph, the curves are noisy and look at the test accuracy. It jumps up and down, which means that with a learning rate of 0.003, the learning process is going too fast. But it is not reasonable to divide the learning rate by 10 or the training would take a long time. We choose to start fast and decay the learning rate exponentially. Following is the formula we used for learning rate: $\text{Learning Rate} = \text{min_learning_rate} + (\text{max_learning_rate} - \text{min_learning_rate}) * \exp(-\text{epoch}/\text{decay_speed})$. Table.1 shows the accuracy and AUC of FTRL with fixed learning rate and with decay learning rate. Obviously, decay learning rate contribute a lot to accuracy.

Different Optimizer

And we use tried different optimizer instead of just using gradient descent optimizer. Following are the results by using different optimizer. Following are the roc curve result for different optimizers we have tried, with other factors remain the same. More detail about ROC are in later evaluation section. All optimizer's AUC is close, but the CV accuracy shows FTRL with decay learning rate has better performance for this model and data set.

Optimizer	Test Accuracy (CV)	Test Standard Deviation	AUC(Area under ROC Curve)
Adam	0.902385	0.00137311	0.96
RMS(With decay = 0.8, mo = 0.1)	0.90055	0.00618432	0.97
FTRL(With fixed learning rate)	0.893578	0.00366974	0.96
FTRL(With decay learning rate)	0.925872	0.00526707	0.97

Table.2 Optimizer evaluating

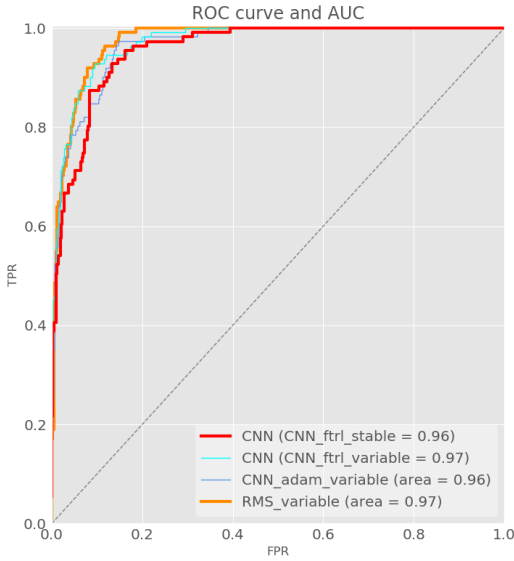


Figure.5 Optimizer ROC

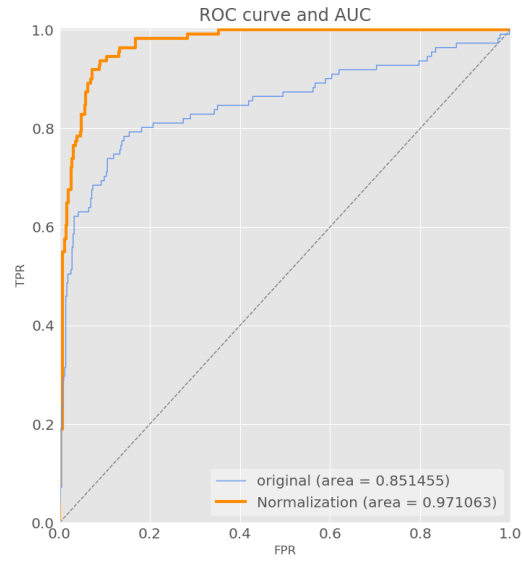


Figure.6 Normalization ROC

Drop Out

In order not to over fit the data, we need drop out. At each training iteration, we drop random neurons from the network. We choose a probability for a neuron to be keep, 0.9. At each iteration of the training loop, we randomly remove neurons with all their weights and biases. Since each neuron reuses the same weights whereas in the fully connected networks seen previously. We could not need to use the dropout on the convolutional layers, but only on the fully-connected layer. Since for convolutional layers, the neurons reuse the same weights, so dropout which effectively works by freezing some weights during one training iteration, would not work on them.

Normalization

Since there is a big scale difference in the maximum pixel in each image, we need to preprocess data. This makes samples have different degree of effect on parameters of the model. Samples with large values has more effect than samples with relative small values. We used normalization on the data to make them all scale in $[-1, 1]$. Below is the comparison between models with normalization and without normalization. Normalization contributes a great improvement of accuracy and AUC. Especially when the model is simple.

With Normalization	Test Accuracy	Standard Deviation	AUC (Area under ROC Curve)
Yes	0.921468	0.00374241	0.971063
No	0.892477	0.00395241	0.851455

Table.3 Normalization evaluating

Bounding

Bounding box extract the main feature of data. The bounding data and original data is provided. This part just compares bounding data and original data in same model to decide which one is used here. Based on the result, bounding box data is used in this model

Data	Accuracy	Standard	AUC(area under the curve)
Original(imgs)	0.889541	0.00728416	0.950098
Partial(bbs)	0.926606	0.00284252	0.9972080

Table.4 Bounding box evaluating

GPU TensorFlow

The GPU TensorFlow is used in our project. This is based on NVIDIA cuda which provide a parallel computing. This greatly improve the speed of training process. It gives us more time to do research on different models and to adjust parameters. A 5 layers CNN with 200 batches, 1000 epoch and 5 folds cross validation spend 4662.46s on CPU but 1026.62s on GPU. GPU gives almost 5 times speed than CPU.

ROC

RoC is an important method to evaluate the performance of binary classifier. It uses true positive rate as y-axis and false positive rate as x-axis. RoC curve shows that, in different threshold, the model gives back the same sensitivity with the same signal inputted. Comparing to the accuracy of model, RoC provides more information to evaluate the performance of binary classifier.

Final Model and Result

Final Model

Our final model is a convolutional neural network with Five-layer. Shown as Figure. 7 Part of important parameters are show below:

Step:1000, batch size 200, learning rate decay: 0.001 – 0.005 decay 2000 Drop out: 0.1

A FTRLOptimizer with decay learning rate is used. Parameters:

Max learning rate: 0.005, Min learning rate: 0.001, Decay speed: 2000

Function: $= \min + (\max - \min) * \exp(-\text{epoch} / \text{speed})$.

Normalized bounding data is used as input.

Cross validation and ROC curve are used to evaluate our model.

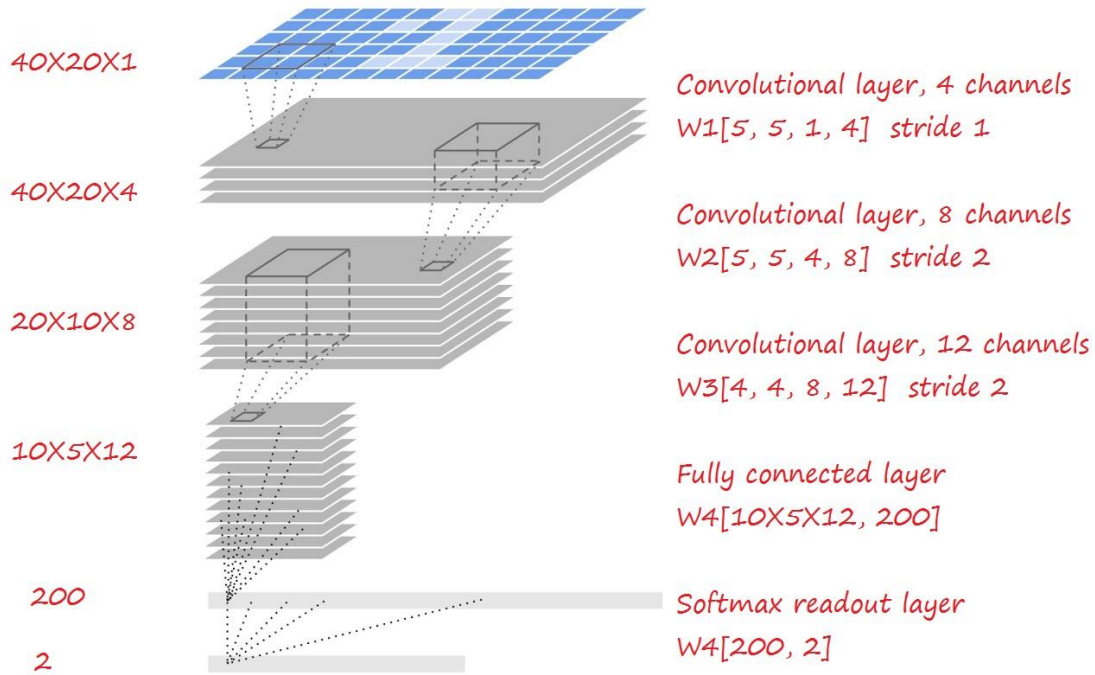


Figure.7 CNN structure

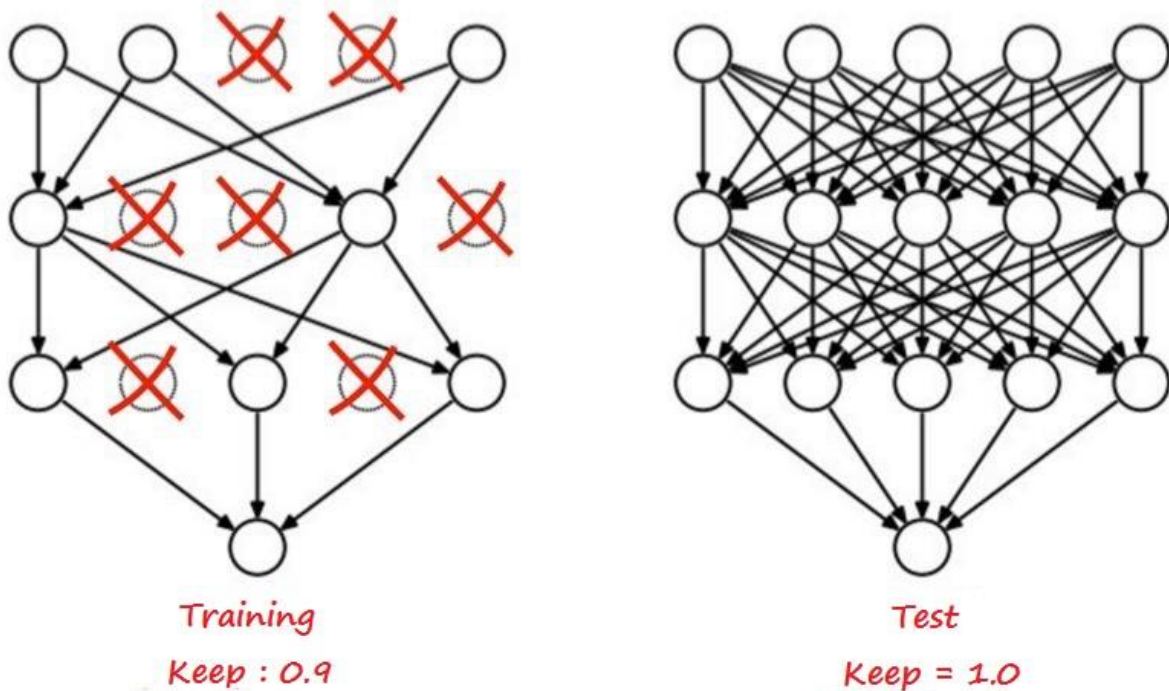


Figure.8 Drop out

Results

Confusion Matrix

Figure.9 shown below is the confusion matrix for our predicted results comparing to the true results. Eleven images with label 0 (bad signal) are predicted to 1 (good signal) and thirty-four images with label 1 (good signal) are predicted to 0 (bad signal). In total there are forty-five mispredictions out of 546 test data. Figure.10 is two examples of mispredictions from label 1 to 0 and from label 0 to 1, from left to right respectively.

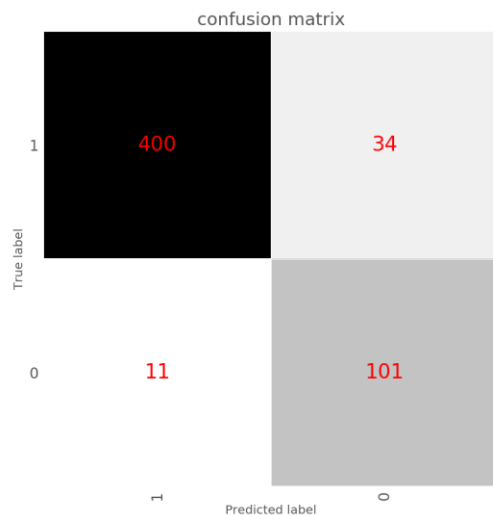


Figure.9 Confusion Matrix

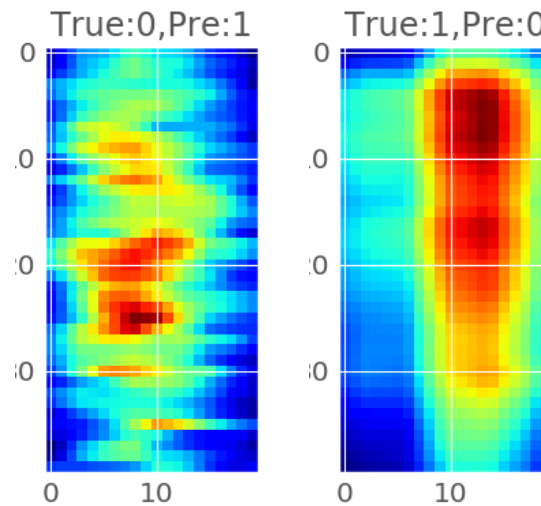


Figure.10 Mispredictions

Accuracy and RoC curve

Table.5 and figure.7 shows the training result of our model. All training means training with all train data (which is 4000 in our project). CV Model 1-5 shows the result of 5 Cross Validation models. All models get almost same AUC and RoC curve. So, Test accuracy is important to evaluate our model. The average score of CV is 0.923. Model 2 has highest accuracy with 0.93.

Model	Test Accuracy (score)	AUC (area under the curve)
All training model	0.917431	0.9698
CV model 1	0.921101	0.9730
CV model 2	0.930275	0.9735
CV model 3	0.921101	0.9707
CV model 4	0.928441	0.9725
CV model 5	0.913761	0.9687
CV mean	0.922936	0.9714

Table.5 Bounding box evaluating

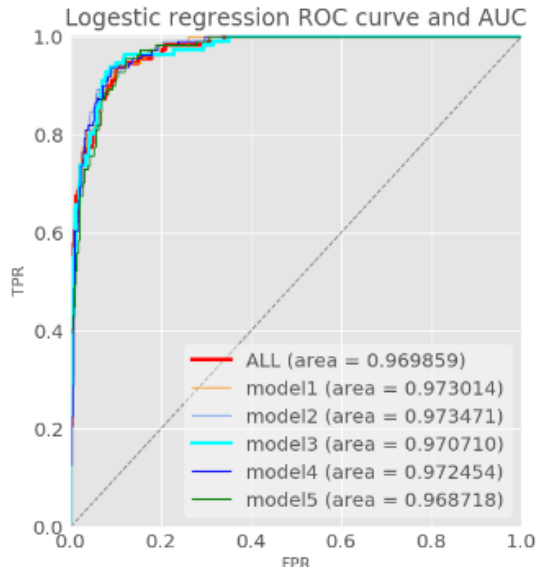


Figure.11 CV model ROC

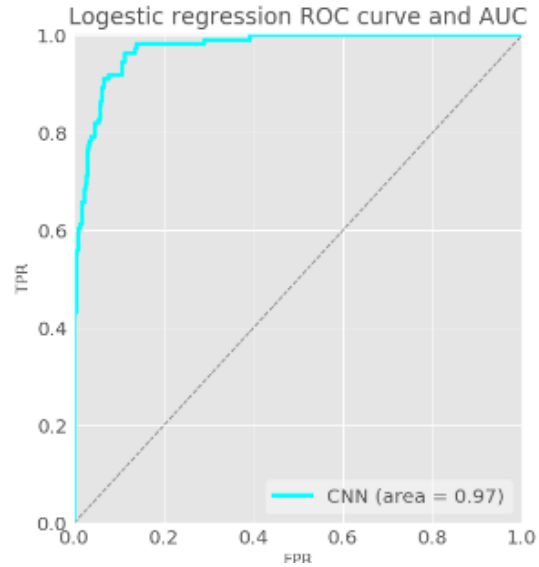


Figure.12 Model 2 ROC

Figure 11 and figure 12 is the ROC and Accuracy curve of model 2. The accuracy increases quickly at beginning and become almost flat after several steps' training. At the same time, loss keeps going down in all. Weights and biases show the distribution of W and B in model.

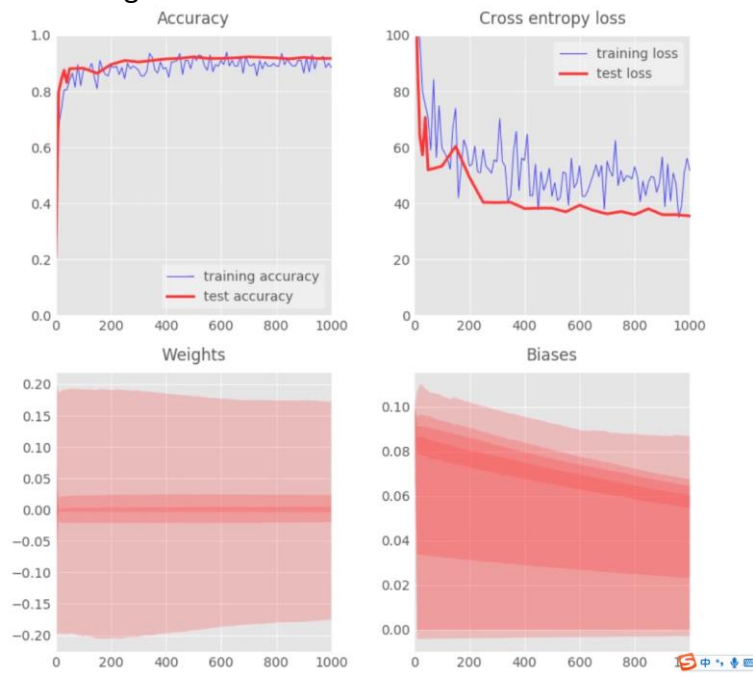


Figure.13 Final model accuracy and loss

Feature Extraction

Based on our final convolutional neural network, following are the features we found for the good and bad signals. Figure 14 and figure 15 are typical original graph of good and bad signals. All the figures with relatively smaller size on the right of the original image are features extracted from first and second layer of convolutional neural network, from left to right and from top to bottom, respectively. We have four features from first layer and eight layers from second layers. Based on the graph, it is obvious that the first two features of the first layer extract the contour of the image and the third features of the first layer extract the main information of the image. Following are some findings from feature extraction:

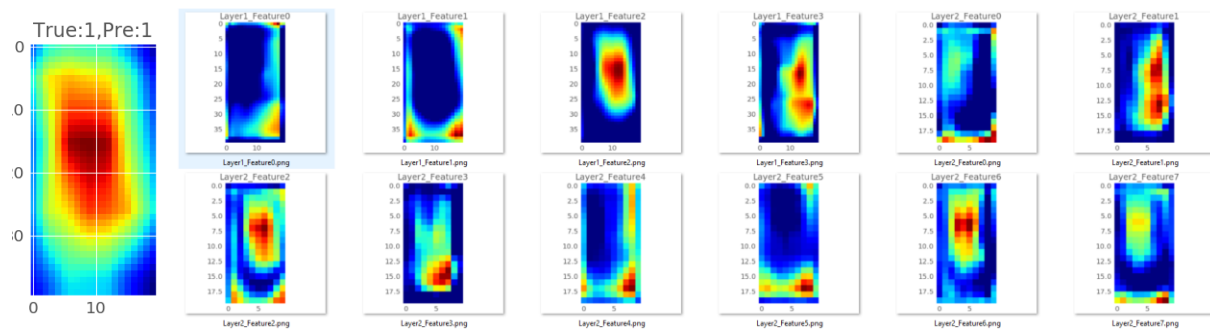


Figure.14 Good signal sample features

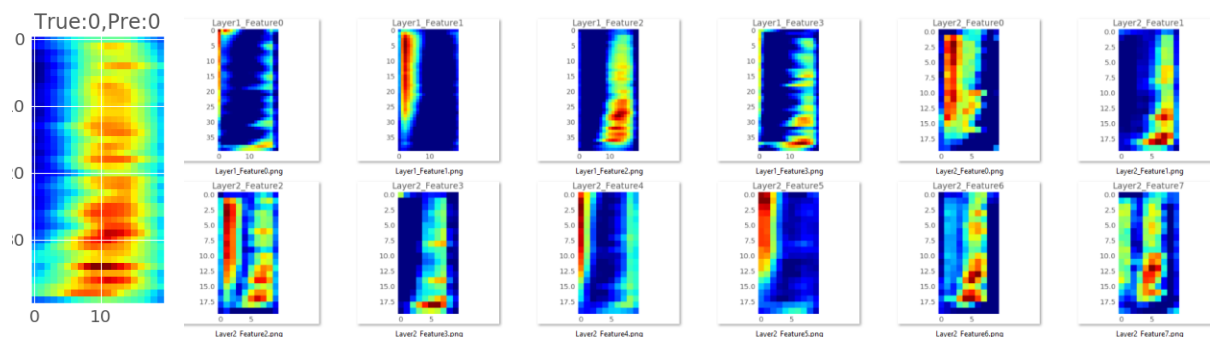


Figure.15 Bad signal sample features

For Good Signals:

1. Identifiable shape (a round shape most of the time) (typically from third feature of first layer)
2. Clear and closed contour, not mixed with border (typically from first and second feature of first layer)
3. Position of the central image, in the middle or upper part of image

For Bad Signals:

1. Not identifiable shape with multi-layers (typically from third feature of first layer)
2. Ambiguous contour, sometimes mixed with border (typically from first and second feature of first layer)
3. Hard to identify a central image, sometimes in the lower part of image

Mislabeled Signals

Since the label (whether a signal is good or bad) is manually created and it is possible that some of the labels are mislabeled. Based on our feature extraction shown above and our understanding about the good and bad signals, we manually corrected eight labels in the training dataset, which corresponding to number 1004, 1005, 1006, 1023, 1032, 1066, 1083 and 1095 in the original dataset. Figure.16 shown below is their original graphs and labels, from top to bottom and left to right, respectively. It turns out that change of these labels does not influence our accuracy by much, which might because we have not find outliers in the images that influence the outcome a lot.

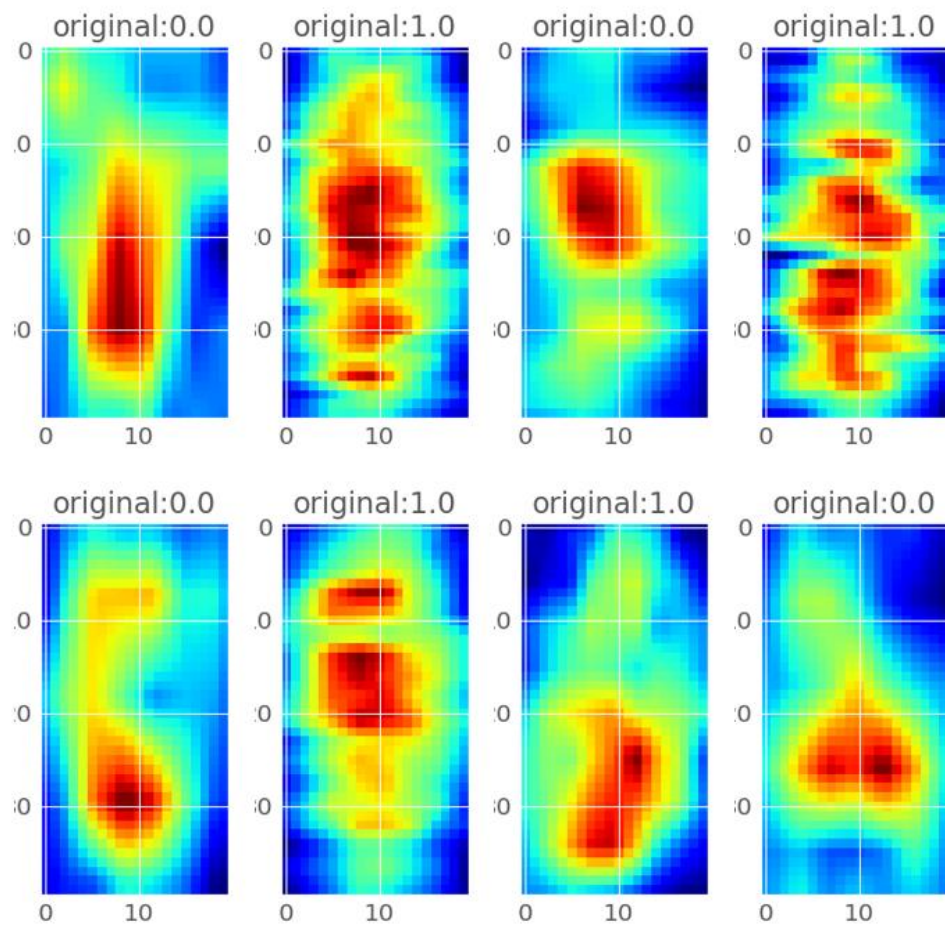


Figure.16 Relabel data

Clustering

Typical Clustering Image

We used the most traditional unsupervised learning method, K-means, to do the clustering. K-means clustering partitions n observations into k clusters following the rule that each observation belongs to the cluster with feature similarity. Following are the typical images in several of the clusters and we found that the images in same cluster is sometimes the same, but some image may look great different even in the same cluster, Like the images in the cluster 11 when $K = 100$. As the number of cluster raising, the difference between images in the same cluster goes down. (Following are only examples of images in several clusters and more details are in the uploaded files.)

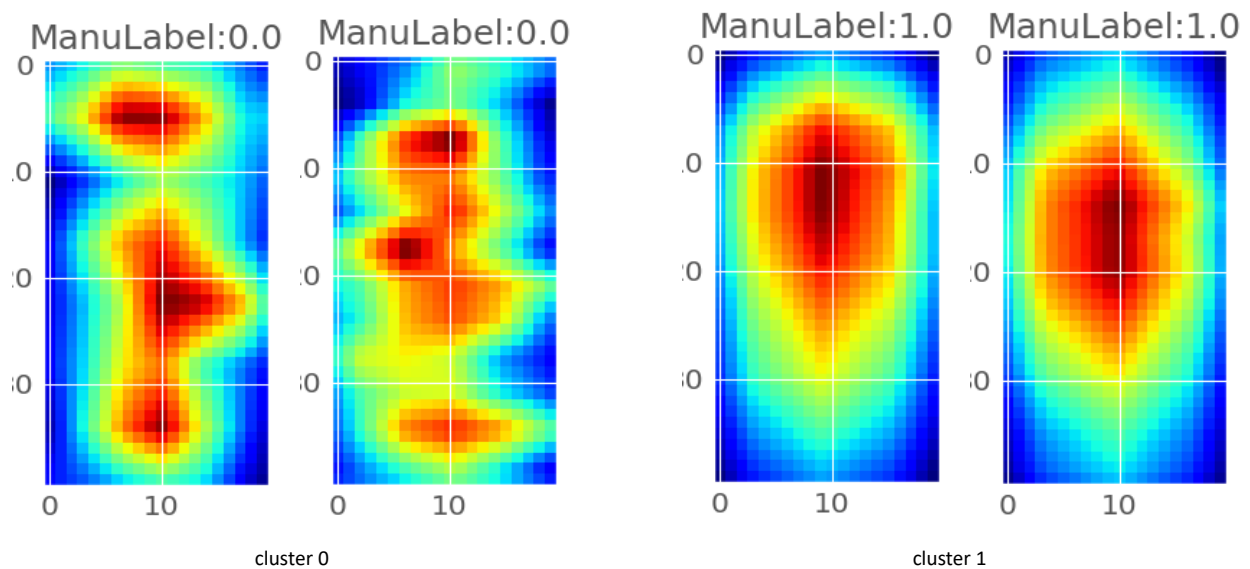


Figure.17 Number of cluster $K = 2$

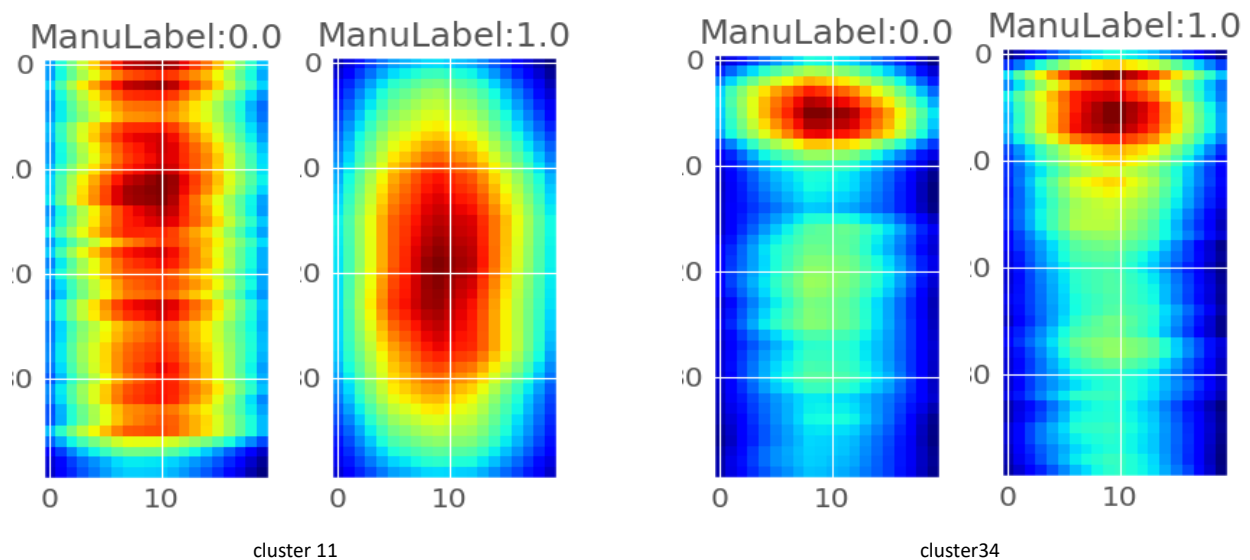


Figure.18 Number of cluster $K = 100$

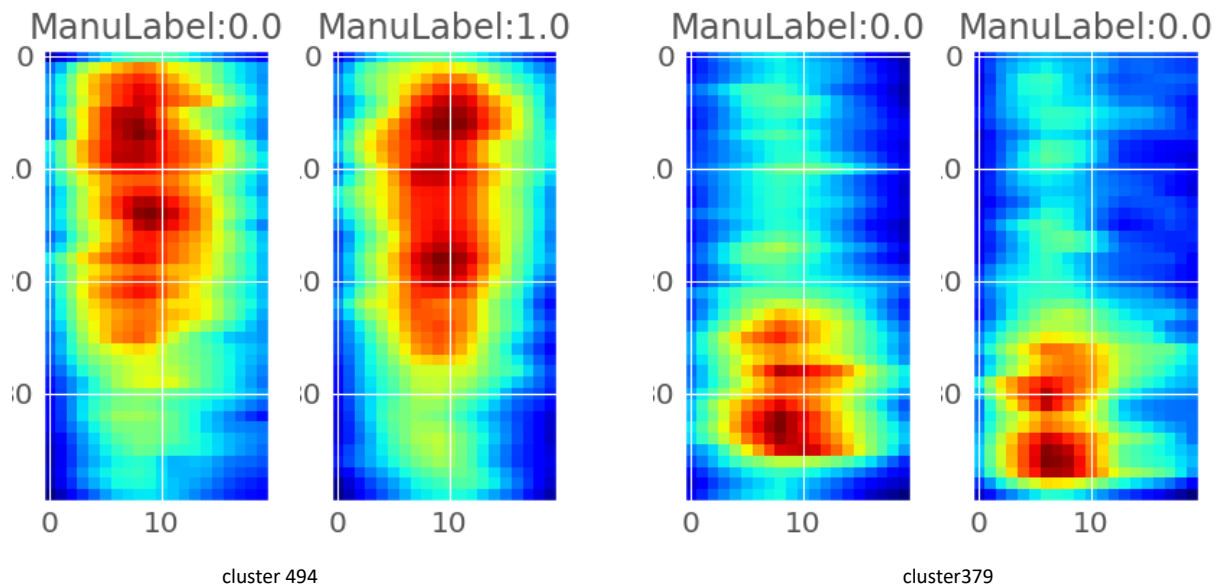


Figure.19 Number of cluster K =500

Silhouette Score

We use silhouette to study the separation distance between the resulting clusters. The silhouette score is a measure of how accurate points are assigned to one cluster. Based on silhouette score, we can assess parameters like number of clusters. Our score for the 2, 100, 200, 500 clusters are 0.9816, 0.4056, 0.3113, and 0.1362 respectively. As the graph shown below, as the number of clusters increases, the silhouette score decreases, which make sense since these images should belong to only two clusters, either good or bad.

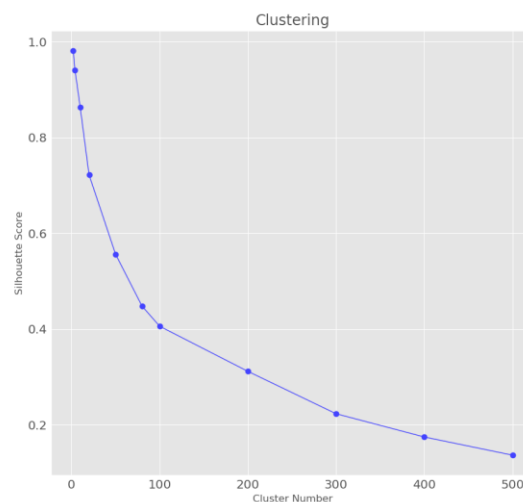


Figure.20 Silhouette Score for different K

Further Discussion

We intended to try another two techniques, batch normalization and grid search.

Batch Normalization

Machine learning methods tends to work better when the input data consist of uncorrelated feature with zero mean and unit variance. When training a neural network, we can preprocess the data before feeding it to the network to explicitly de correlate its features; this will ensure that the first layer of the network process the data with a standard and uncorrelated distribution. However, the activation at deeper layers of the network will no longer be de correlated and will no longer have zero mean or unit variance since they are output form earlier layers in the network. Even worse, during the training process the distribution of features at each layer of the network will shift as the weights of each layer are updated, which as hypothesized by the author of [3], the shifting distribution of features inside deep neural networks may make training deep networks more difficult. To solve this problem, [3] proposes to insert batch normalization layers into the network, to have learnable shift and scale parameters for each feature dimension.

Batch Normalization solve the problem that changes in model parameters during learning change the distributions of the outputs of each hidden layer. The later layers need to adapt to these changes during training. But this can limit the representational power of the layer, therefore, we allow the network to undo the batch normalizing transform by multiplying by a new scale parameter γ and adding a new shift parameter β . β and γ are learnable parameters.

Grid Search

When choosing the parameter of model, the most expensive cost is people's focus. To reduce the focus on parameter, we introduce Grid Search algorithm. This algorithm can train model based on the given range of parameter. Then, the algorithm will choose the best one according to the score of their cross validation.

Due to time constrain, we haven't finished these two techniques.

Reference:

- [1]. <https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#0>
- [2]. https://github.com/martin-gorner/tensorflow-mnist-tutorial/blob/master/mnist_4.2_batchnorm_convolutional.py
- [3]. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift"
- [4]. <https://github.com/cthorey/CS231/blob/master/assignment2/BatchNormalization.ipynb>
- [5]. <https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#9>
- [6] <https://www.datascience.com/blog/k-means-clustering>
- [7] http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

