COMP5212 Machine Learning Project 2 Report

Jianhao JIAO, 20475718, jjiao@ust.hk

April 6, 2018

1 Data Set

1. CNN Model: `data_classifier_train.npz`(40000 images) and `data_classifier_test.npz`(4000 images)

2. CAE Model: `data_autoencoder_train`(70000 images) and `data_autoencoder_eval.npz`(1000 images)

2 Notation and abbreviation

1. c: stop criterion
2. $step_{train}$: training step
3. η: learning rate
4. α: momentum
5. a_{train}: classification accuracy on training datasets
6. a_{test}: classification accuracy on testing datasets
7. a_{eval}: classification accuracy on evaluation datasets
8. l_{train}: log loss on training datasets
9. l_{test}: log loss on testing datasets
10. l_{eval}: log loss on evaluation datasets
11. τ_{train}: training time
12. CNN: convolutional neural network
13. CAE: convolutional autoencoder model

Figure 1: 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).

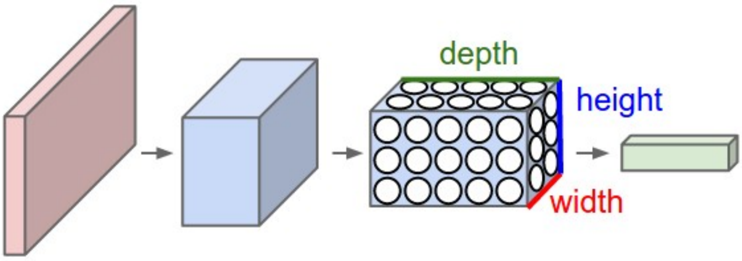
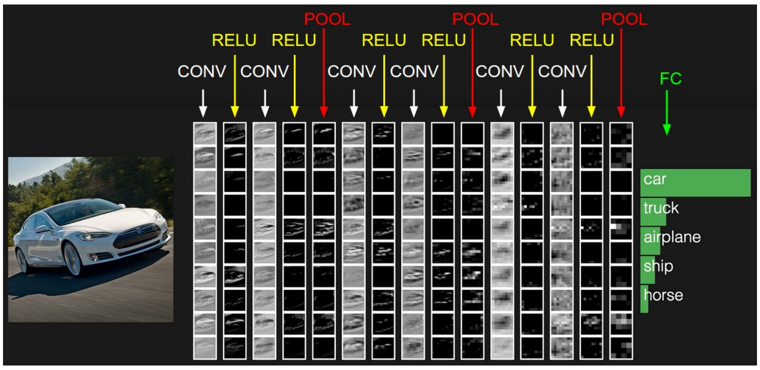


Figure 2: The activations of an example ConvNet architecture. The initial volume stores the raw image pixels (left) and the last volume stores the class scores (right). Each volume of activations along the processing path is shown as a column. Since it's difficult to visualize 3D volumes, we lay out each volume's slices in rows. The last layer volume holds the scores for each class, but here we only visualize the sorted top 5 scores, and print the labels of each one. The full **web-based demo** `http://cs231n.stanford.edu/` is shown in the header of our website. The architecture shown here is a tiny VGG Net, which we will discuss later.



3 CNN

1. Principle

   (a) CNNs are a special type of neural networks for processing data with a known grid-like topology, e.g.

      i. Spatial data: 2 spatial dimensions

      ii. Spatiotemporal data: 1 spatial dimension, 1 temporal dimension

   (b) Corss-Correlation:

$$s[i,j] = (x * w)[i,j] = \sum_{n=-M}^{M} \sum_{n=-N}^{N} x[i+m, j+n]w[m,n]$$

   (c) Fig.1 and Fig.2 are two examples of CNN.[1]

[1] http://cs231n.github.io/convolutional-networks/

Table 1: Equipment Description

Hardware	Performance
CPU	Intel Core i7-7700K clocked at 4.2 GHz $\times$ 8
GPU	Nvidia GeForce GTX 1080 Ti/PCIe/SSE 2
Memory	15.6 GiB
OS	Ubuntu 16.04_64-bit

(d) Mini-batch SGD with momentum algorithm: The gradient update follows the below equation[2]:

$$m_t = \alpha \times m_{t-1} + \eta \times \bigtriangledown_{\theta_{t-1}} f(\theta_{t-1})$$

$$\Delta\theta_t = -m_t$$

where η is the learning rate, α is the momentum factor, $\bigtriangledown_{\theta_t} f(\theta_t)$ is the gradient at t.

Algorithm 1 Mini-batch SGD with momentum algorithm

set $\chi^{(1)}, ..., \chi^{(N)} = \{\text{images, labels}\}$, b(batch size), α, η, **w**,
c(stopping criteria), $t_{\mathbf{max}}$(maximum training step);
repeat
  each batch $\chi^{(\ell_0)}, \chi^{(\ell_1)}, ..., \chi^{(\ell_0+b)}$;
  repeat
    predictions = **CNN**($\chi^{(\ell_0)}, \chi^{(\ell_1)}, ..., \chi^{(\ell_0+b)}$, **w**);
    loss = **Cross_entropy**(predictions, $\chi^{(\ell_0)}, \chi^{(\ell_1)}, ..., \chi^{(\ell_0+b)}$);
    $\Delta\mathbf{w}_t$ = **MomentOptimizer**(η, α, loss);
    $\mathbf{w}_t$ = $\mathbf{w}_{t-1} + \Delta\mathbf{w}_t$;
    $t = t+1$;
  until ($\Delta loss < c$) $\|$ ($t > t_{\mathbf{max}}$);
until $\ell_0 + b = N$;

Algorithm 2 Moment optimizer

MomentOptimizer(η, α, loss):
g_t = **Compute_Gradient**(loss);
$\eta_{tensor}, \alpha_{tensor}$ = **Convert_to_tensor**(η, α);
m_t = -$\alpha_{tensor} \times m_{t-1} + \eta_{tensor} \times g_t$;
update = m_t;
return update

2. Experiment

   (a) Equipment Description: Please refer to Table 1

   (b) CNN architecture: Following the project description, we implement the below architecture: 3, 4

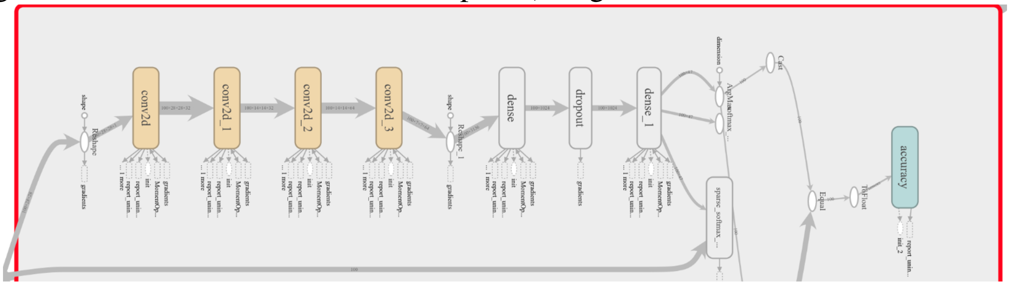
   (c) Parameters tuning using cross validation techniques: In this section, the invariant and variant parameters and settings will be shown below. The variant items will be changed

[2] https://en.wikipedia.org/wiki/Stochastic_gradient_descent

Figure 3: The CNN architecture description.

Layer Name	Description	# Filters	Filter Size	Stride
Input	Input image	NA	NA	NA
Conv1	ConvLayer	32	3×3	1
Conv2	ConvLayer	32	5×5	2
Conv3	ConvLayer	64	3×3	1
Conv4	ConvLayer	64	5×5	2
FC	1024-unit fc	NA	NA	NA
Output	Prediction	NA	NA	NA

Figure 4: The CNN architecture description(using **Tensorboard Visulization Tool**).



for building different CNN model. As the cross validation technique, we use the **Hold-out Method**[3], where we randomly split 80% training data to **train the model** and 20% training data to **evaluate the model**.

i. **Invariant parameters:** The Table2 are invariant in the further model building.

ii. **Variant parameters:** In the further steps, we will change the **Learning rate** and **Momentum** observer the model's performance. Table3 show the candidate parameters which we will use later.

[3]https://en.wikipedia.org/wiki/Cross-validation_(statistics)

Table 2: The invariant parameters and setting.

Maximum Training Steps	Batch Size	Convergence Criteria	Initializer of weights
30000*5	100	0.001	Xavier Initializer

Table 3: Some candidate parameters.

Learning Rate(η)	Momentum(α)
0.01	0.1
0.01	0.01
0.001	0.1
0.001	0.01
0.0001	0.1
0.0001	0.01

Figure 5: Loss over times setting **LR=0.01** and **MM=0.1**

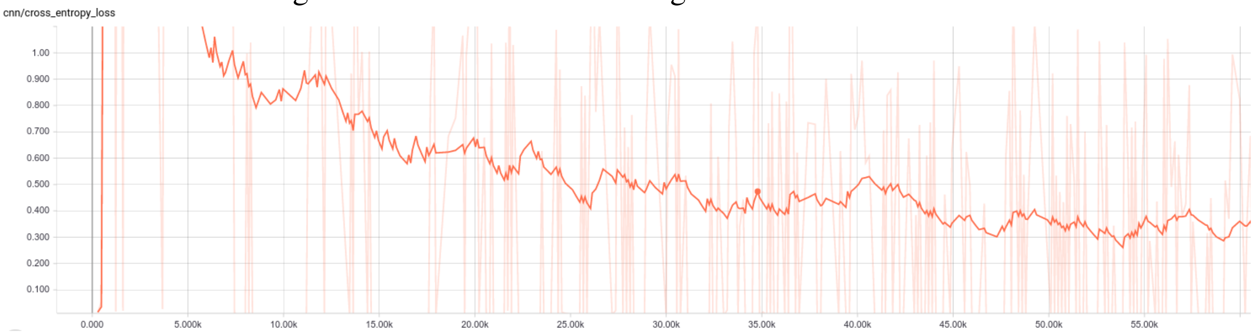
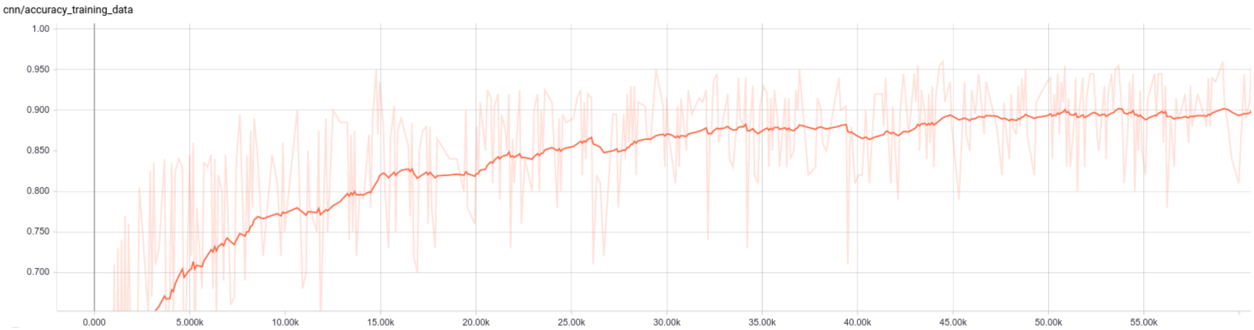


Figure 6: Accuracy over times setting **LR=0.01** and **MM=0.1**



iii. **Paramenter tunning:** setting **LR=0.01** and **MM=0.1**:

A. Loss over times(on training data): Please refer to Fig.5

B. Accuracy over times(on training data): please refer to Fig.6

C. Result(finish training): please refer to Table.4

Table 4: Result on the built CNN model.

τ_{train}	$step_{train}$	l_{train}	l_{eval}	a_{train}	a_{eval}
1102.32s	61021	0.2518	0.1.2624	94.21%	84.16%

Figure 7: Loss over times setting **LR=0.01** and **MM=0.01**

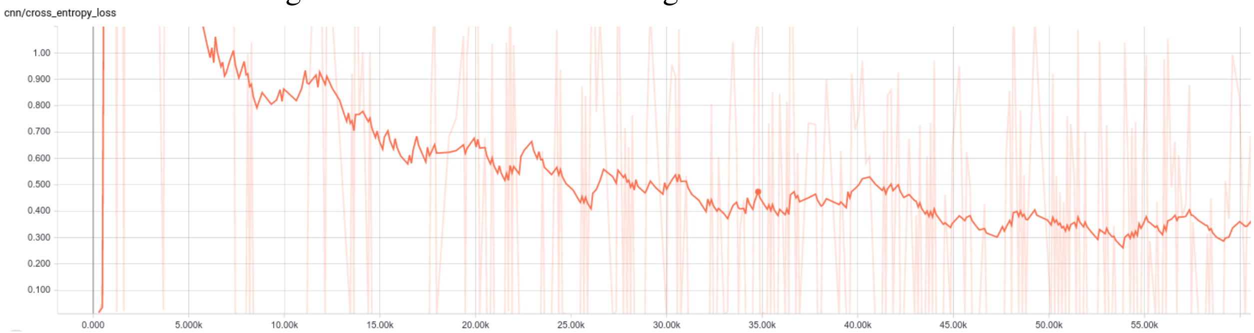
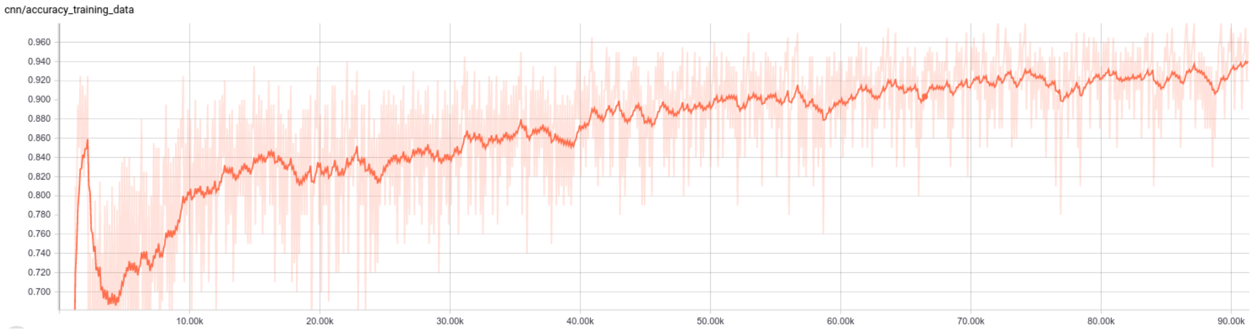


Figure 8: Accuracy over times setting **LR=0.01** and **MM=0.01**



iv. **Paramenter tunning:** setting **LR=0.01** and **MM=0.01**:

A. Loss over times(on training data): please refer to Fig.7

B. Accuracy over times(on training data): please refer to Fig.8

C. Result(finish training): please refer to Table.5

Table 5: Result on the built CNN model.

τ_{train}	$step_{train}$	l_{train}	l_{eval}	a_{train}	a_{eval}
1508.50s	91368	0.4081906	1.0913395	91.45%	84.32%

Figure 9: Loss over times setting **LR=0.001** and **MM=0.1**

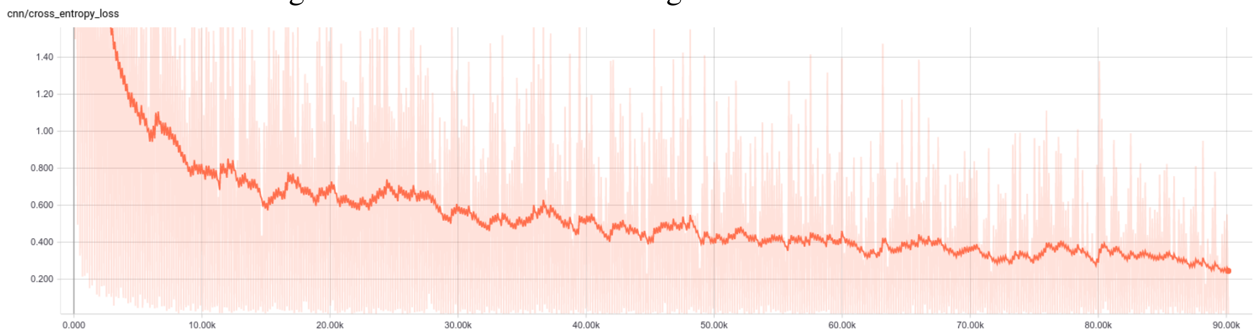
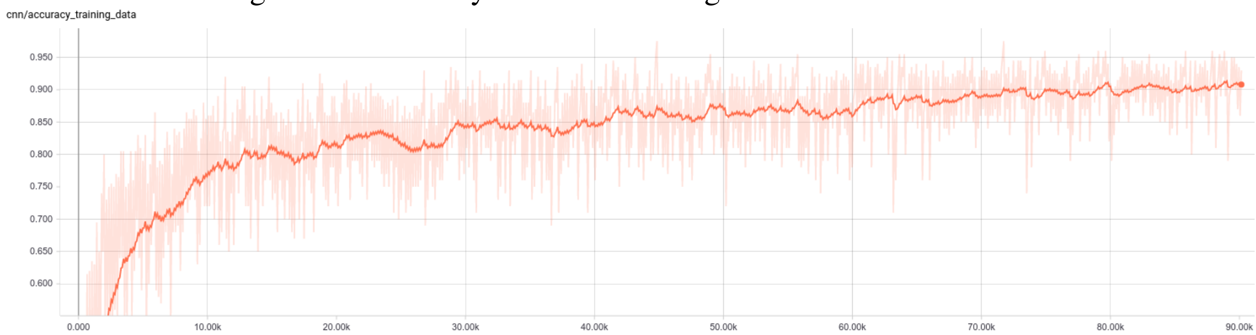


Figure 10: Accuracy over times setting **LR=0.001** and **MM=0.1**



v. **Paramenter tunning:** setting **LR=0.001** and **MM=0.1**:
   A. Loss over times(on training data): please refer to Fig.9
   B. Accuracy over times(on training data): please refer to Fig.10
   C. Result(finish training): please refer to Table.6

Table 6: Result on the built CNN model.

τ_{train}	$step_{train}$	l_{train}	l_{eval}	a_{train}	a_{eval}
1445.81s	90240	0.5016	1.0841	88.96%	83.05%

Figure 11: Loss over times setting **LR=0.001** and **MM=0.01**

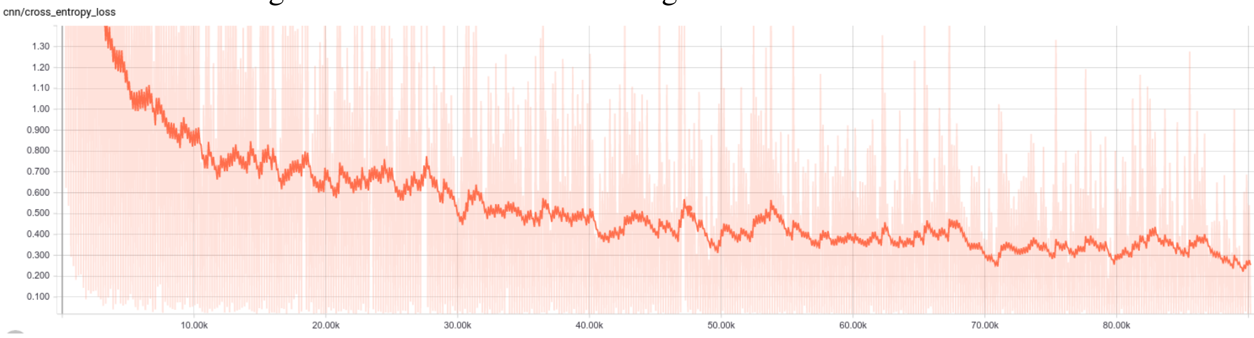
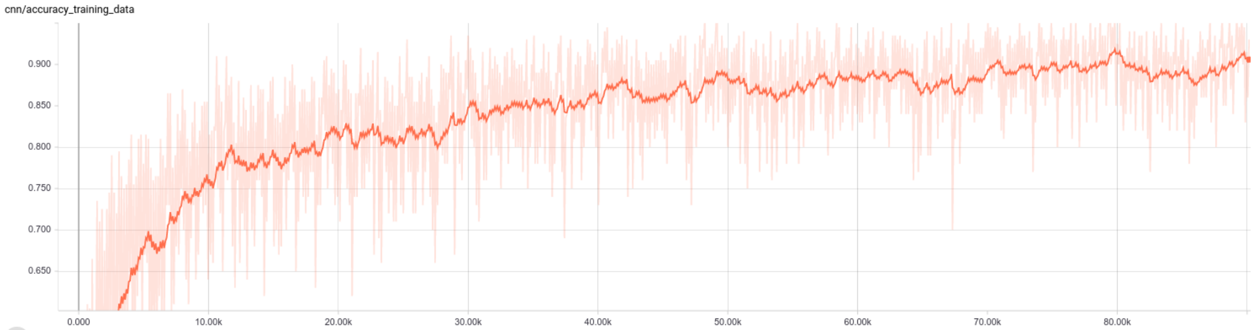


Figure 12: Accuracy over times setting **LR=0.001** and **MM=0.01**



vi. **Paramenter tunning:** setting **LR=0.001** and **MM=0.01**:

A. Loss over times(on training data): please refer to Fig.11

B. Accuracy over times(on training data): please refer to Fig.12

C. Result(finish training): please refer to Table.7

Table 7: Result on the built CNN model.

τ_{train}	$step_{train}$	l_{train}	l_{eval}	a_{train}	a_{eval}
1467.87s	90240	0.5432	1.0958	88.71%	82.79%

Figure 13: Loss over times setting **LR=0.0001** and **MM=0.1**.

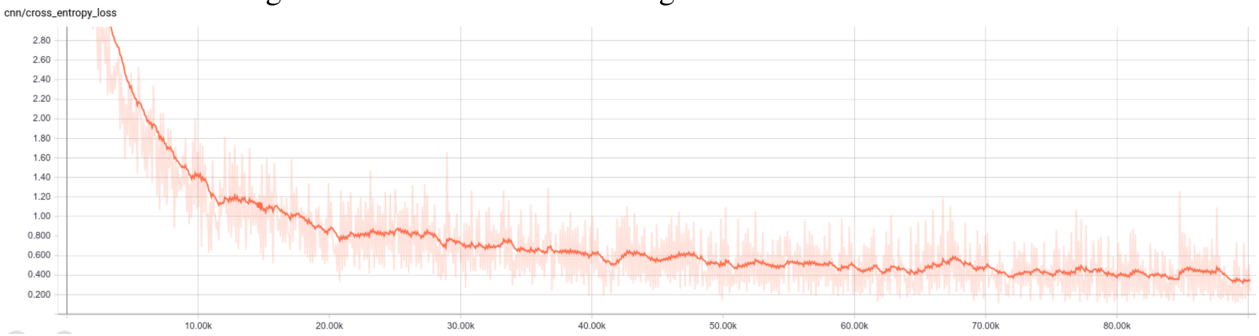
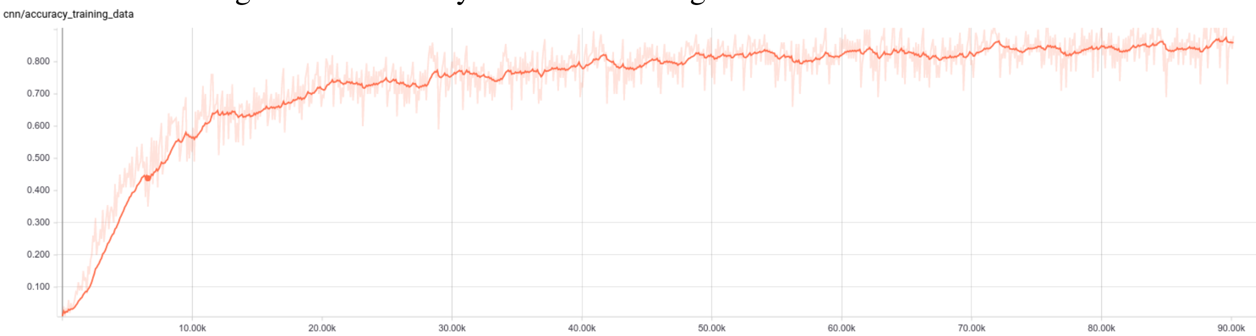


Figure 14: Accuracy over times setting **LR=0.0001** and **MM=0.1**.



vii. **Paramenter tunning:** setting **LR=0.0001** and **MM=0.1**:
   A. Loss over times(on training data): Please refer to Fig.13
   B. Accuracy over times(on training data): please refer to Fig.14
   C. Result(finish training): Please refer to Table.8

Table 8: Result on the built CNN model.

τ_{train}	$step_{train}$	l_{train}	l_{eval}	a_{train}	a_{eval}
1406.01s	90240	0.4232039	0.6377447	86.35%	81.96%

Figure 15: Loss over times setting **LR=0.0001** and **MM=0.01**

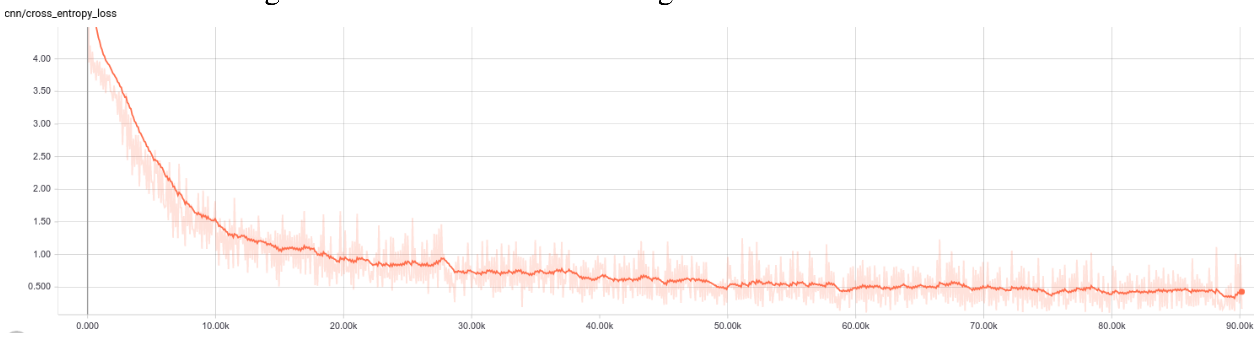
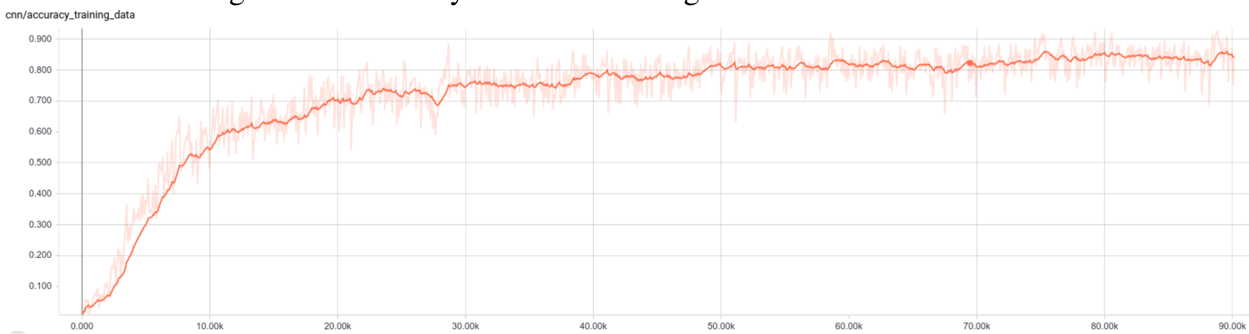


Figure 16: Accuracy over times setting **LR=0.0001** and **MM=0.01**



viii. **Paramenter tunning:** setting **LR=0.0001** and **MM=0.01**:
   A. Loss over times(on training data): please refer to Fig.15
   B. Accuracy over times(on training data): please refer to Fig.16
   C. Result(finish training): please refer to Table.9

Table 9: Result on the built CNN model.

τ_{train}	$step_{train}$	l_{train}	l_{eval}	a_{train}	a_{eval}
1486.05s	90240	0.4271182	0.6352878	85.95%	81.67%

Figure 17: Loss over times setting **LR=0.0001** and **MM=0.1** with whole training dataset.

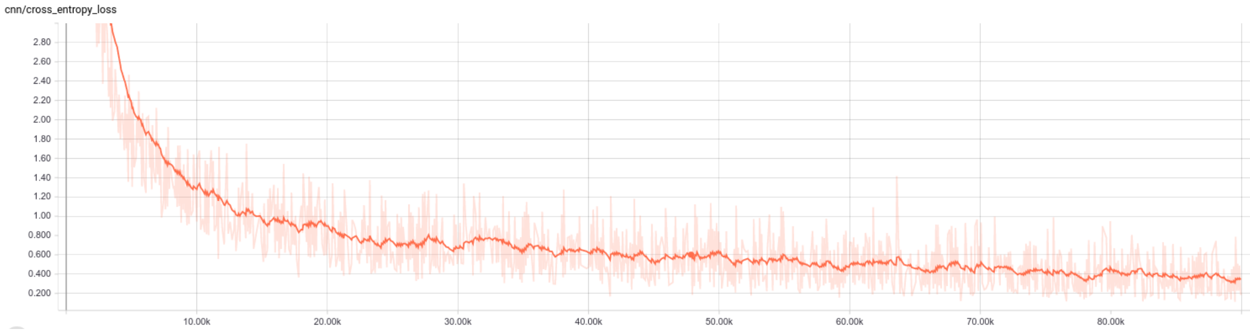
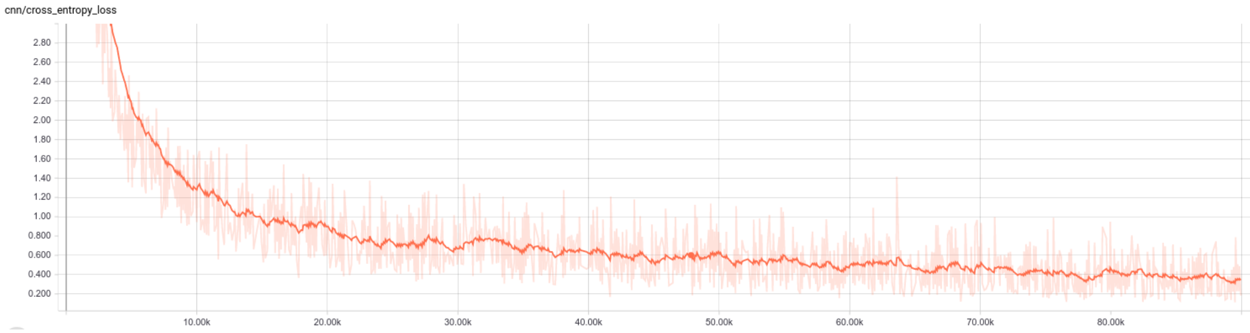


Figure 18: Loss over times setting **LR=0.0001** and **MM=0.1** with whole training dataset.



(d) **Accuracy and loss on the training and test sets:** Comparing the $l_{train}, l_{eval}, a_{train}, a_{eval}$, we prefer to choose **LR=0.0001** and **MM=0.1** as a pair of parameters. And with such parameters, we use 100% training data to train the CNN model and then test it with both 100% training data and 100% testing data.

i. Loss over times: Please refer to Fig.17

ii. Accuracy over times: Please refer to Fig.18

iii. Result: Please refer to 10

Table 10: Accuracy and loss on the training and test sets.

τ_{train}	$step_{train}$	l_{train}	l_{test}	a_{train}	a_{test}
1574.02s	90000	0.40926594	0.643690	86.29%	81.35%

Figure 19: An example of a network structure of deep autoencoder.

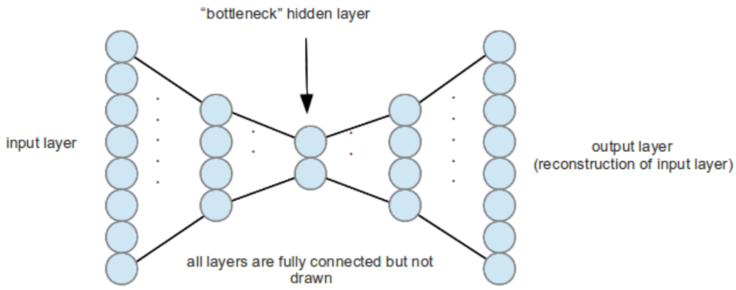


Figure 20: The CAE architecture description.

Layer Name	Description	# Filters	Filter Size	Stride
Input	Input image	NA	NA	NA
Enc1	ConvLayer	32	5×5	2
Enc2	ConvLayer	64	5×5	2
Enc3	ConvLayer	2	3×3	1

4 CAE

1. Principle

   (a) Deep Autoencoder: A deep autoencoder is a multilayer extension of the simple autoencoder by using multiple hidden layers. The middle layer still acts as a bottleneck. Fig.19 is an example of the network structure of a deep autocoder.

2. Experiment

   (a) Equipment Description: Please refer to Table 1

   (b) CAE architecture: Following the project description(Fig,20), we implement the below architecture: 21(using `tensorboard` to visulize the network structure).

   (c) Parameters tuning using cross validation techniques: In this section, the invariant and variant parameters and settings will be shown below. The variant items will be changed for building different CAE model. As the cross validation technique, we use the **Hold-out Method**[4] where we randomly split 80% training data to train the model and 20% training data to evaluate the model.

      i. **Invariant parameters:** Parameters in the table11 are invariant(same in CNN model) in the further model building.

      ii. **Variant parameters:** In the further steps, we will change the **Learning rate** and **Momentum factor** observer the model's performance. The table 12 show the candidate parameters which we will use later.

[4] https://en.wikipedia.org/wiki/Cross-validation_(statistics)

Table 11: The invariant parameters and setting.

Maximum Training Steps	Batch Size	Convergence Criteria	Initializer of weights
None	20	0.001	Xavier Initializer

Figure 21: The CAE architecture description(using **Tensorboard Visulization Tool**).

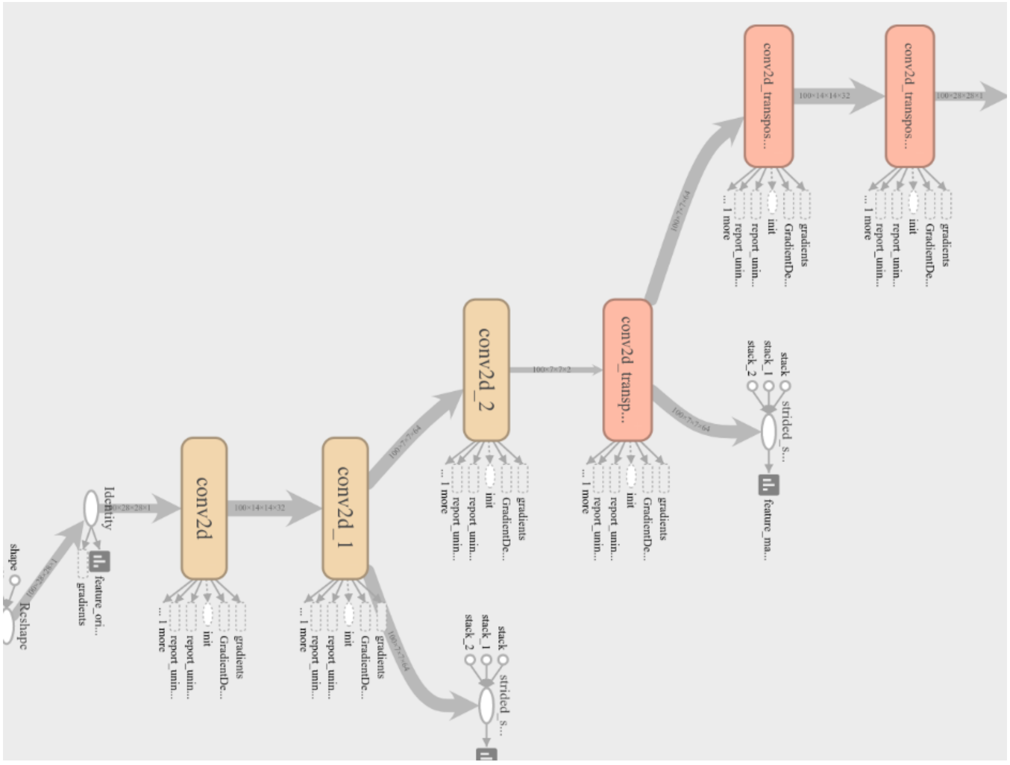


Table 12: Some candidate parameters.

Learning Rate(η)	Momentum(α)
0.01	0.2
0.05	0.1
0.001	0.5
0.005	0.3
0.005	0.5
0.008	0.2

Figure 22: Loss over times setting **LR=0.01** and **MM=0.2**.

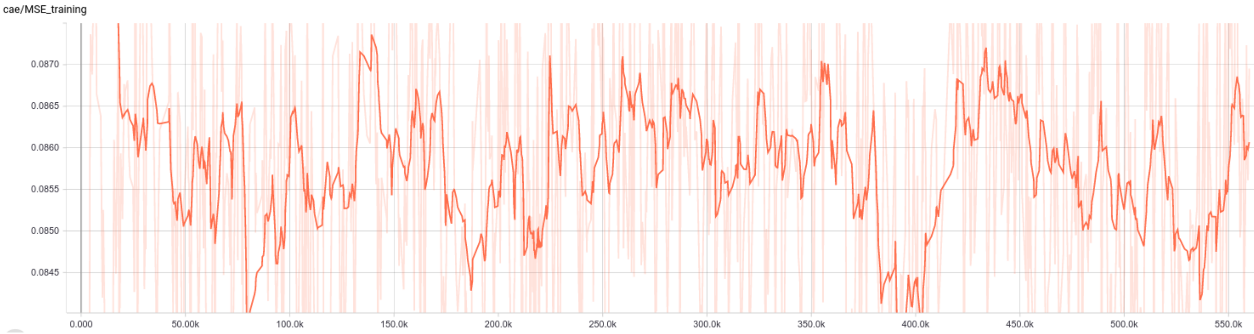
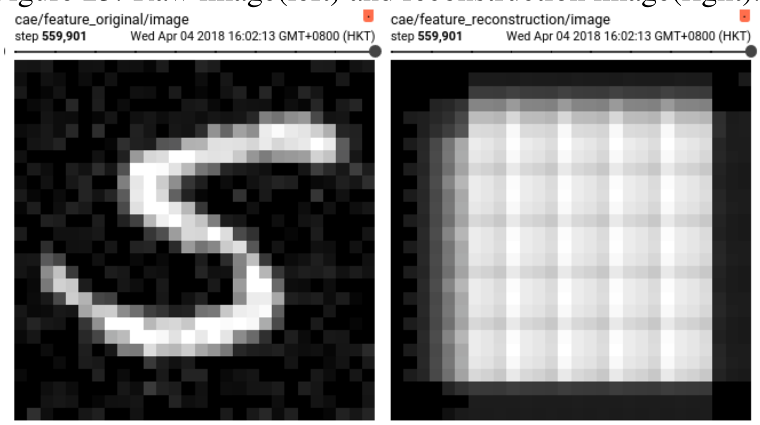


Figure 23: Raw image(left) and reconstruction image(right).



iii. **Paramenter tunning:** setting **LR=0.01** and **MM=0.2**:

A. Loss over times(on training data): Please refer to Fig.22

B. Result: Please refer to Fig.23 for the reconstruction images and Table.13 for training report.

Table 13: Training steps, training time and MSE on training and evaluation datasets.

τ_{train}	$step_{train}$	l_{train}	l_{eval}	Note
2273.02s	302000	0.08902343	0.086723	could not reconstruct

Figure 24: Loss over times setting **LR=0.05** and **MM=0.1**.

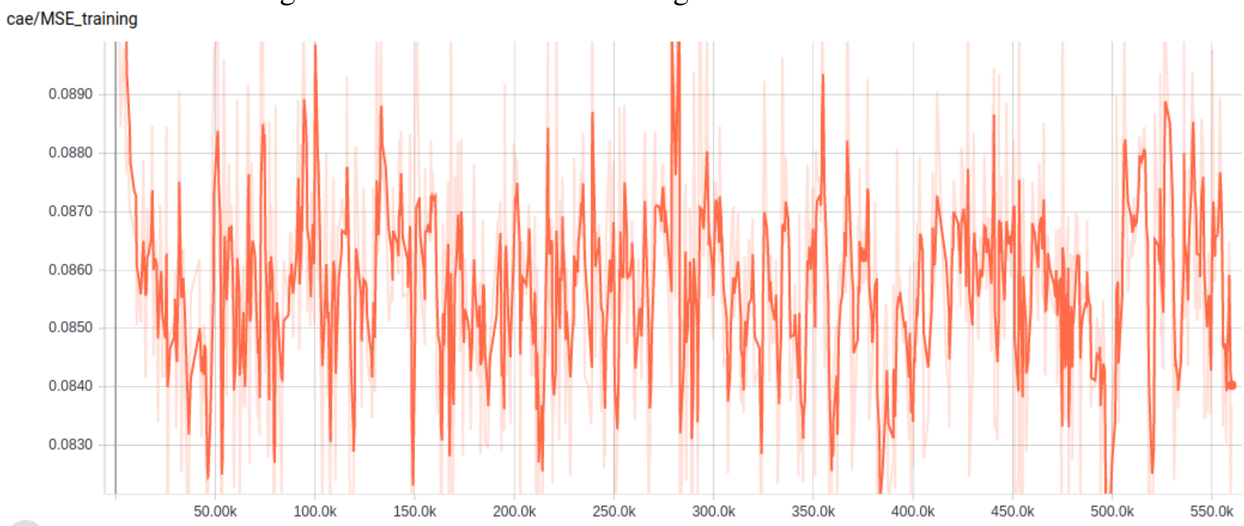
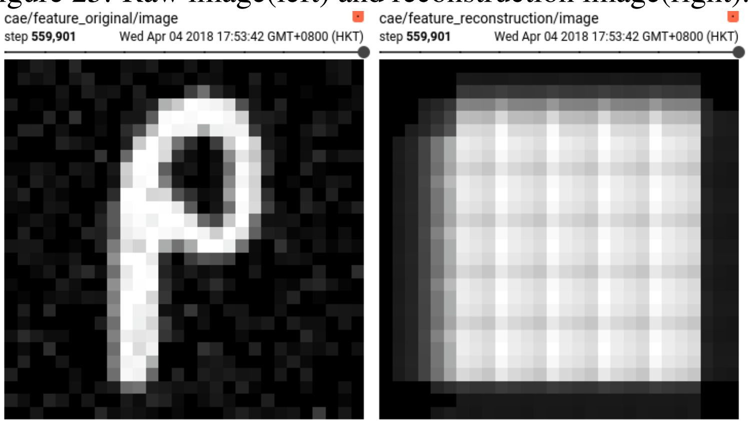


Figure 25: Raw image(left) and reconstruction image(right).



iv. **Paramenter tunning:** setting **LR=0.05** and **MM=0.1**:

A. Loss over times(on training data): Please refer to Fig.24

B. Result: Please refer to Fig.23 for the reconstruction images and Table.14 for training report.

Table 14: Training steps, training time and MSE on training and evaluation datasets.

τ_{train}	$step_{train}$	l_{train}	l_{test}	Note
2843.07s	560000	0.0857261	0.0854096	could not reconstruct

Figure 26: Loss over times setting **LR=0.001** and **MM=0.5**.

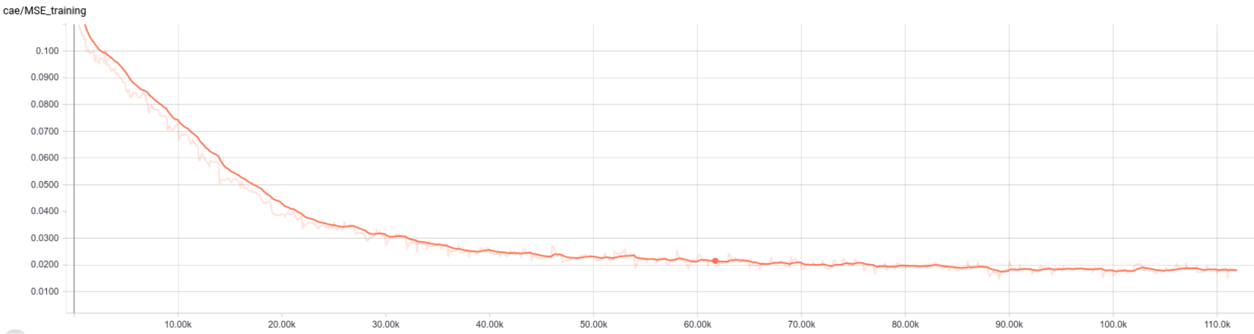
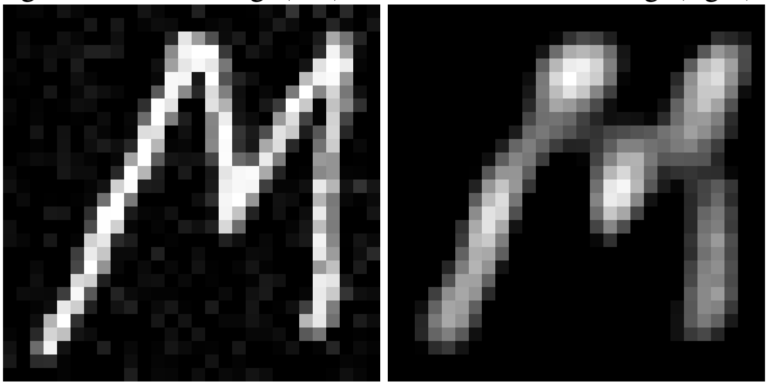


Figure 27: Raw image(left) and reconstruction image(right).



v. **Paramenter tunning:** setting **LR=0.001** and **MM=0.5**:
   A. Loss over times(on training data): Please refer to Fig.26
   B. Result: Please refer to Fig.27 for the reconstruction images and Table.15 for training report.

Table 15: Training steps, training time and MSE on training and evaluation datasets.

τ_{train}	$step_{train}$	l_{train}	l_{test}	Note
1469.03s	110200	0.0298475	0.024722	could reconstruct

Figure 28: Loss over times setting **LR=0.005** and **MM=0.3**.

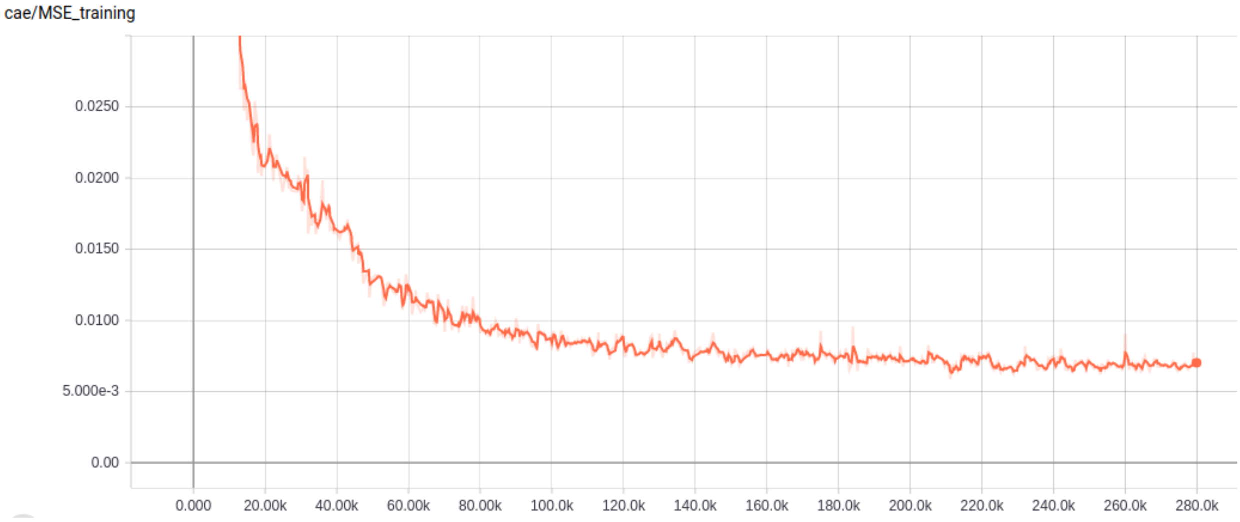


Table 16: Training steps, training time and MSE on training and evaluation datasets.

τ_{train}	$step_{train}$	l_{train}	l_{test}	Note
24310.3s	280000	0.0086423	0.0086213	could reconstruct

vi. **Paramenter tunning:** setting **LR=0.005** and **MM=0.3**:
   A. Loss over times(on training data): Please refer to Fig.28
   B. Result: Please refer to Fig.29 for the reconstruction images and Table.16 for training report.

Figure 29: Raw image(left) and reconstruction image(right).

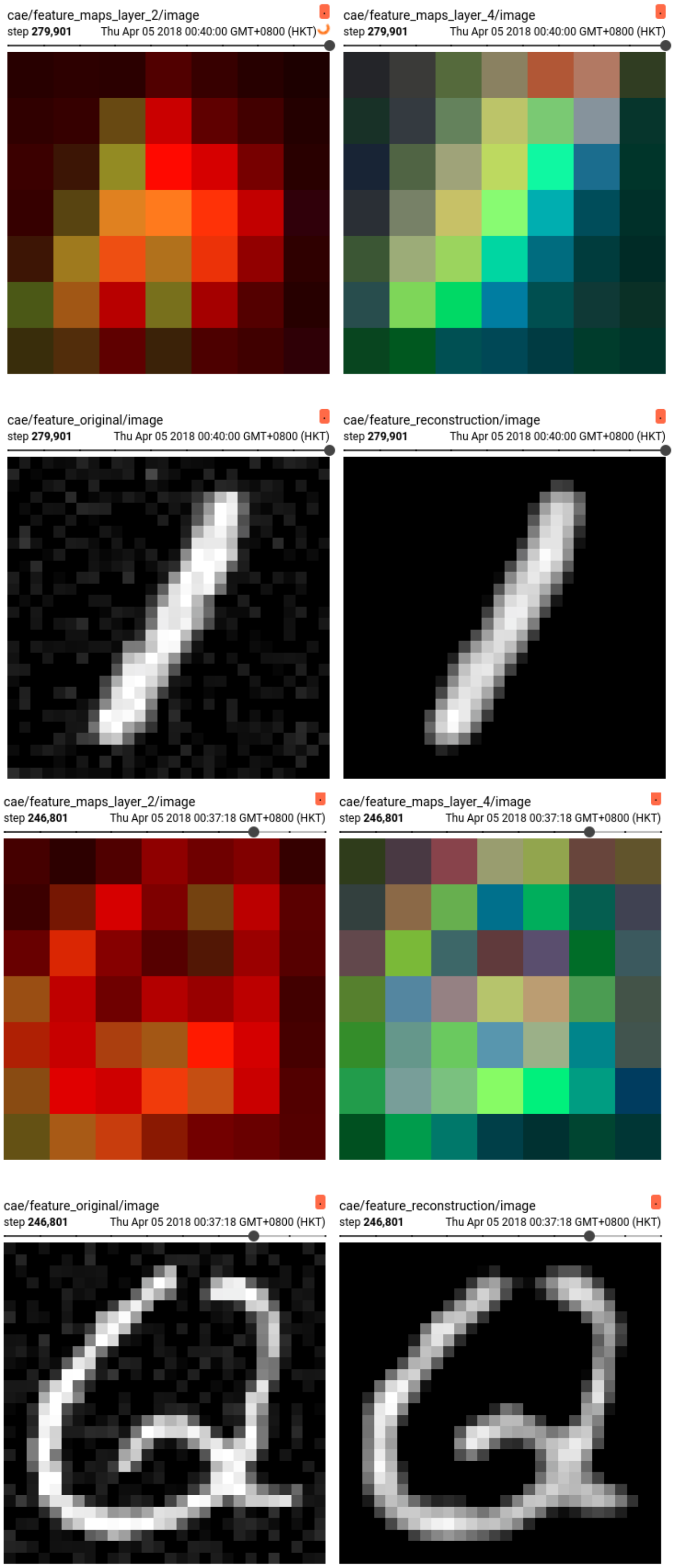


Figure 30: Loss over times setting **LR=0.005** and **MM=0.5**.

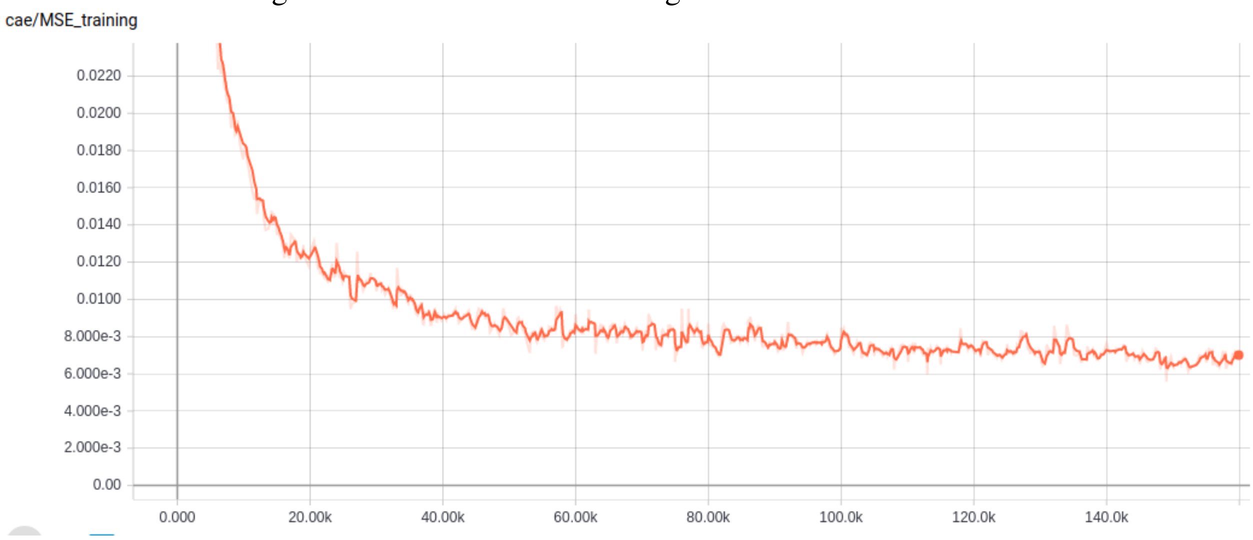


Table 17: Training steps, training time and MSE on training and evaluation datasets.

τ_{train}	$step_{train}$	l_{train}	l_{test}	Note
14231.2s	165000	0.008321	0.0091324	could reconstruct

vii. **Paramenter tunning:** setting **LR=0.005** and **MM=0.5**:

A. Loss over times(on training data): Please refer to Fig.30

B. Result: Please refer to Fig.31 for the reconstruction images and Table.17 for training report.

Figure 31: Raw image(left) and reconstruction image(right).

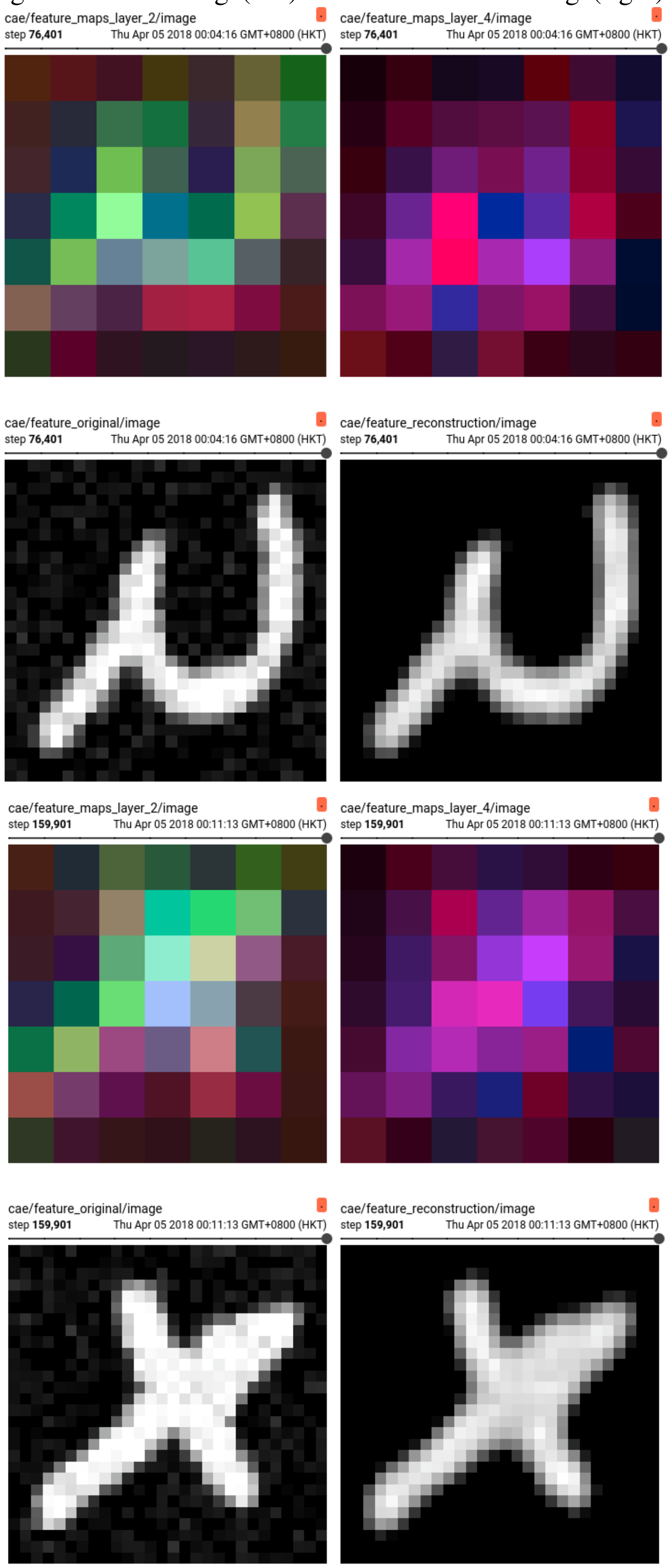


Figure 32: Loss over times setting **LR=0.008** and **MM=0.2**.

Table 18: Training steps, training time and MSE on training and evaluation datasets.

τ_{train}	$step_{train}$	l_{train}	l_{test}	Note
27312.1s	280000	0.0094231	0.00986213	could reconstruct

viii. **Paramenter tunning:** setting **LR=0.008** and **MM=0.2**:

A. Loss over times(on training data): Please refer to Fig.32

B. Result: Please refer to Fig.33 for the reconstruction images and Table.18 for training report.

Figure 33: Raw image(left) and reconstruction image(right).

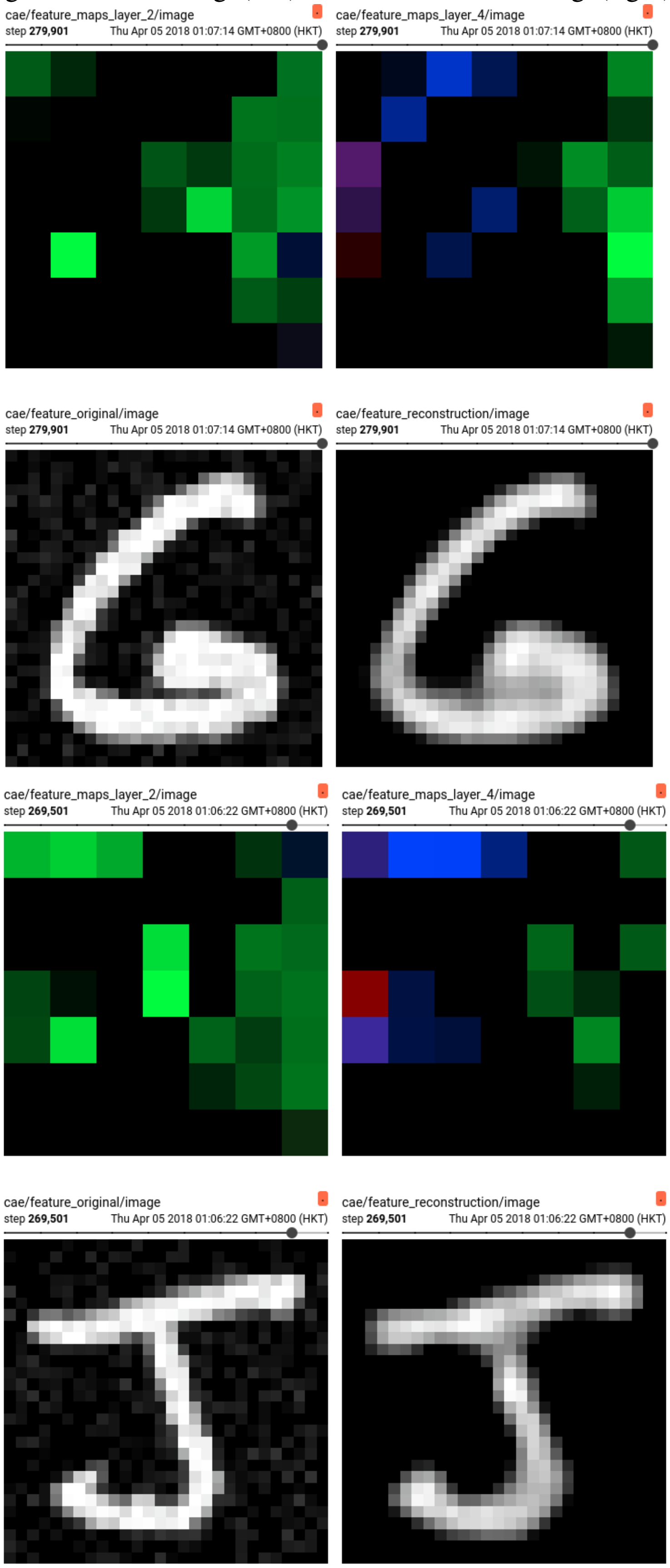


Figure 34: Loss over times setting **LR=0.005** and **MM=0.5** with whole training dataset.

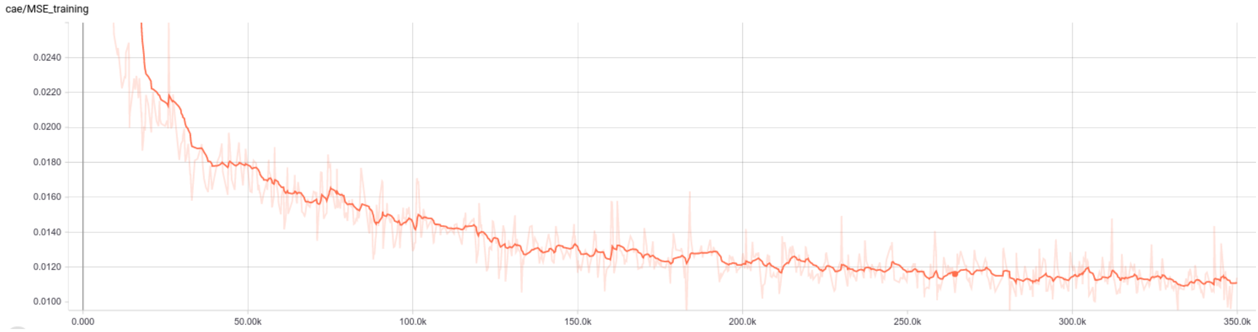


Table 19: Training steps, training time and MSE on training and evaluation datasets.

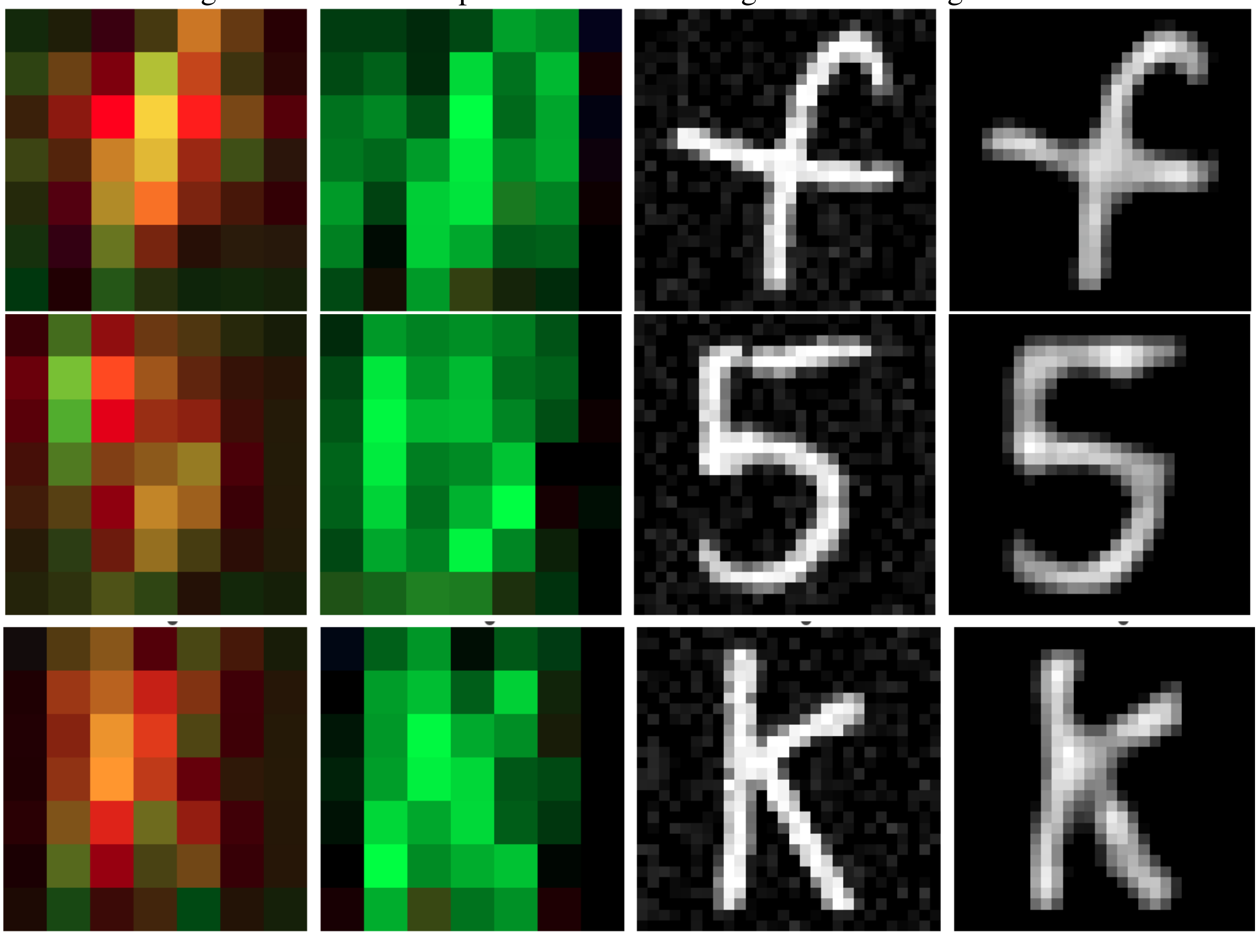
τ_{train}	$step_{train}$	l_{train}	l_{test}	Note
2843.06s	560000	0.0857261	0.011086	could reconstruct

ix. **Feature Maps on evaluation dataset:** By comparing the l_{train}, l_{eval} and observing the clarity of feature maps and reconstruction images among different parameter selections, we consider **LR=0.005** and **MM=0.5** as a pair of parameters has the best performance(Although the **LR=0.005** and **MM=0.5** pair has lower MSE, but it is unstable such that it can not reconstruct images at multi times). With such parameters, we use 100% training data to train the CAE model and then evaluate it (plot some feature maps and reconstruction images) with the provided evaluation data.

A. Loss over times: Please refer to Fig.34.

B. Result: Please refer to 35 to for the reconstruction images and feature maps on testing sets and Table 19 for the training report.

Figure 35: Feature maps and restruction images on the testing datasets.



5 Some useful notes

1. As far as I know, `tensorflow` provides two kinds of network design pipeline examples and fucntions: `tf.nn` and `tf.layer`. I prefer the latter one because its design is more abstract with higher level.

2. The CAE model training is sensitive to the selection of **learning rate** and **momentum factor**. Small learning rate(< 0.01) and large momentum(> 0.1) are recommended.

3. Tensorboard is a well-designed and user-friendly visulization tool.