CS420 Machine Learning
Final Project

716030210013 Mengyun Chen
716030210014 Yuge Zhang

June 14, 2018

Summary

1 Introduction

The requirement of the final project is to build a standard classifier for an "non-standard version" of MNIST dataset. It is also required to use at least **three** different approaches to complete the task.

In this report, we will first discuss the dataset, including how it is different from the standard version of MNIST dataset and what kind of information is useful in classification. Then we try some classic and simple approaches to address the issues. Some of them worked out; others didn't. Anyway, we will reach performance that is good enough (about 97% to 98%) with fewer than 100 lines of code and minutes of computing power on a laptop. For the last part, we will further implement the famous Variational Auto-Encoder, and some of its variations, as discussed in class. This neural network is capable of "reconstruction", which will further demonstrate the distribution of the data. The benchmark result and comparison between models that have been used in all these experiments are attached at the end of the report.

2 Analysis

2.1 Dataset technical specifications

Training sample size 60000.

Features 45×45.

Color Grey. Integer range in $[0, 255]$.

Labels Integer in $[0, 9]$.

Test sample size 10000.

2.2 Difference from the standard version

At first, we ran our tests on the standard version of MNIST, because somehow we couldn't download the dataset from the given address and therefore we assumed the dataset was exactly the same as the standard one. It wasn't until someone was starting to write the report that we discovered this little trick: **modified version**.

After reading the dataset and observe a few examples, we conclude that the modified version is actually modified in two ways:

1. The image is randomly extended with black blocks.
2. Some random noise has been added to the image.

Note that we can use some preprocessing techniques to safely crack the first issue. That is, to locate the average / median position of the "white" pixels, and compute a 32×32 wrapper around this center. 32×32 is a little bit larger than the standard 28×28, but this is fine and it should leave some space for possible deviations.

Using 80 as threshold (pixels with values greater than 80 are considered as white), the program takes about 300 seconds to finish with `numpy` implementation.

Let's call this process Centering. Figure 2.1 shows how this procedure affects the image.

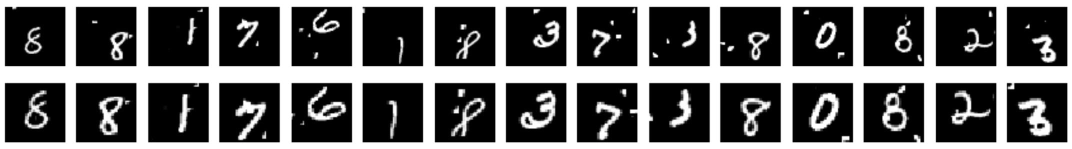


Figure 2.1: The upper row is for raw image. The bottom one is the processed.

2.3 Hints from the legendary Homework 3

This seems to be a standard classifcation problem, like many we have handled before which we can deal with using algorithms like SVM, kNN and similar. Homework 3 did offer us some hint on processing such tasks with sklearn library. We could simply copy the classifier "template" and modify some parameters and achieve not-bad results.

Thresholding Not sure about the exact reason, but the intuition is that rounding the colors to only two: black and white, should help. (Maybe because it will help SVM build the gap since margin is always difficult for grey.) This is another preprocess step: **thresholding**. However, it turns out that this step is actually a trade-off. Benchmark results show that this intuition might be true for time usage, but not for accuracy.

PCA with whitening In homework 3, when dealing with the deep learning image recognition dataset, the unsupervised technique that uses PCA with whitening is used as a pre-step for the classification, and will contribute a lot to the performance: not only accuracy but also time usage. We will combine this technique with almost every method in this subsection.

kNN, SVM, MLP We tried these three models with different combinations of methods mentioned above. kNN and SVM are very slow, and the performance is hardly satisfying. MLP is way much faster and the performance is not bad.

There is not much to discuss since the codes are all intuitively short. For now, we will not go into details, but later will discuss the comparisons between these traditional methods (maybe except MLP), and deep learning.

All experiment results are available in section 4.

2.4 Hints from Tensorflow tutorial

Too many people have done too much research into MNIST dataset, so that the Tensorflow's tutorial for beginners is about MNIST...

Without GPU / TPU acceleration and little Tensorflow programming experience, training with Convolutional Neural Network seems to be both time consuming and unrewarding. With a modified version (only modified on the input data set size) of the CNN in the tutorial, we can only get accuracy less than 95% with over an hour of training. Just for visualization purposes, we plot the cross entropy loss function with respect to training steps. See Figure Figure 2.2.

A re-implementation Actually, after we finish the part with Variational Auto Encoder, we re-implemented the CNN using Tensorflow low-level API. We omitted the dropout layer, for a mysterious belief that overfitting would accelerate the learning with almost little damage. The network structure is shown in Figure 2.3.

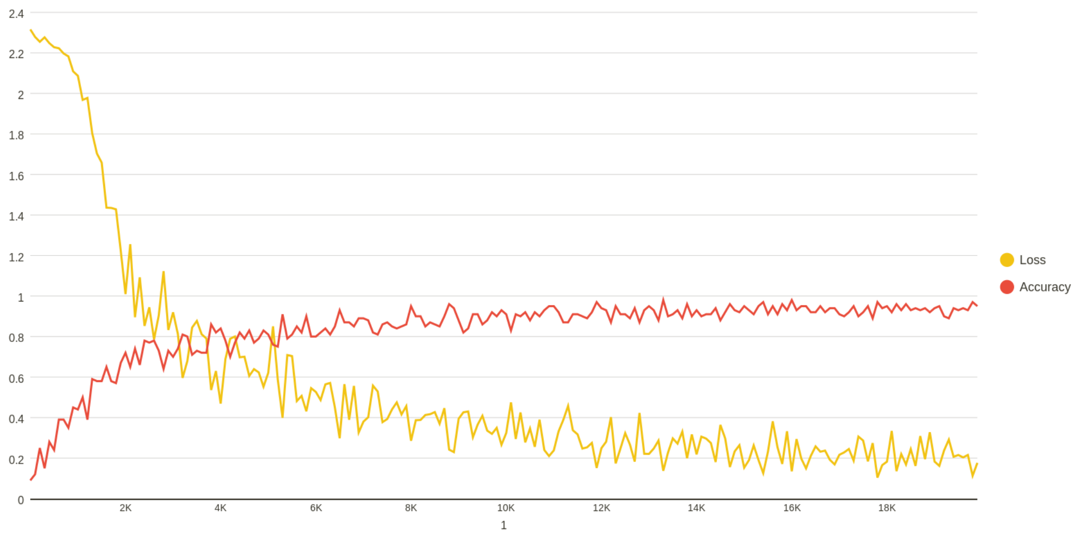


Figure 2.2: CNN diagram. Generated by Tensorboard

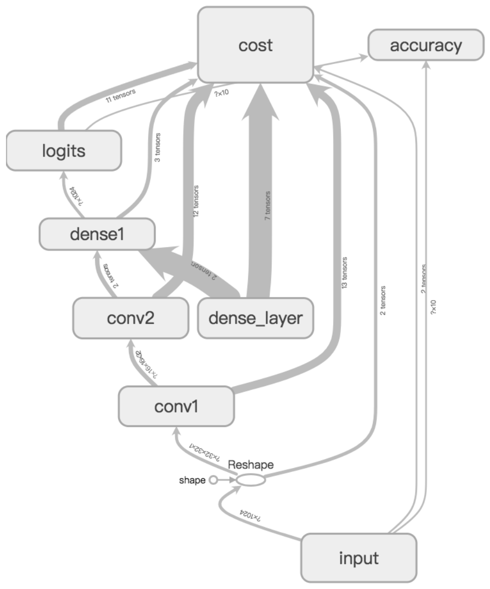


Figure 2.3: CNN diagram. Generated by Tensorboard

The experiment result with respect to training epochs is shown in Table 1[1].

Epoch	Loss	Time	Accuracy
1	0.196677	57.9	0.9723
2	0.055729	60.2	0.9802
3	0.028606	60.7	0.9832
4	0.018365	61.4	0.9843
5	0.015604	62.2	0.9851
6	0.012914	60.5	0.9865
7	0.009439	60.9	0.9893
8	0.008812	60.9	0.99
9	0.007666	61.0	0.9887
10	0.006197	60.4	0.9882
11	0.007844	60.9	0.9877
12	0.005575	60.9	0.9896
13	0.005449	61.2	0.988
14	0.004095	61.4	0.9877
15	0.005644	61.6	0.9858
16	0.005891	61.2	0.9913
17	0.002485	61.9	0.9918
18	0.005556	61.1	0.9905
19	0.004799	61.5	0.9896
20	0.00405	61.7	0.9905

Table 1: CNN varying epoch times

It can be shown in the experiment result that the network has improved very little after epoch 3. For a deeper insight, we reduce the sample number of training data and see what effect it takes on the final result. The evaluation size is the same as before: 10000 samples. Now we have Table 2.

#samples	Epoch 1	Epoch 2	Epoch 3	Final [2]	Total time
1000	0.6431	0.7545	0.8087	6: 0.8374	18.8
2000	0.744	0.8617	0.9034	6: 0.9128	24.6
4000	0.8752	0.9251	0.9357	11: 0.9549	68.6
8000	0.9331	0.9474	0.9476	8: 0.9647	80.2
16000	0.9492	0.9597	0.9662	10: 0.9733	178.7
32000	0.9648	0.9729	0.9805	8: 0.9827	266.6
All	0.9723	0.9802	0.9832	20: 0.9918	1219.5

Table 2: CNN varying training sample number

The convolution net is amazingly efficient and can reach about 93% percent accuracy with sample size equal to the test sample size.

[1]This test uses the same machine as in the benchmark. See the benchmark section for details.

[2]Epoch number and the final accuracy. Training quits if accuracy hasn't improved during last two epochs or it has reached 20.

3 Experiment with VAE

3.1 Variational Auto Encoder

3.1.1 Introduction

Variational Auto Encoder, aka, VAE, is an unsupervised deep learning technique, that enables us to project the data into a latent space, and reconstruct our input using a sample in the latent space.

As VAE is unsupervised, it might be not intuitive to use VAE in a supervised labeling task. The reasons why VAE is introduced in this project are as follows:

1. A step for dimension reduction; a substitute for PCA. Prepare for possible following supervised learning steps.
2. Noise reduction. The input contains a lot of noise. Hopefully we can use reconstruction to remove those noise.
3. Generate cool / fancy visualized facts about MNIST dataset.

3.1.2 Technical details

Not sure about this part, but it seems important that we "redefine" some symbols here to make sure everything is clear.

We now have input $\boldsymbol{x}$. What VAE is trying to do is to find a latent variable $\boldsymbol{z}$ and reconstruct $\boldsymbol{x'}$. To make this happen, VAE builds two neural networks: encoder and decoder. For the decoder, VAE also assumes that $p(\boldsymbol{z}|\boldsymbol{x})$ is a multidimensional guassian distribution with mean $\mu(\boldsymbol{x})$ and variance $\Sigma(\boldsymbol{x})$; and the decoder part can be think of as a function that maps input x into $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. For the decoder, we randomly sample a $\boldsymbol{z}$ from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. And we apply a symmetircal network to compute $\boldsymbol{x'}$. Now comes the magical part, the loss function is computed as follows[3]:

$$\textsc{ReconstrLoss} = \boldsymbol{x}\log(\boldsymbol{x'}) + (1-\boldsymbol{x})\log(1-\boldsymbol{x'}) \tag{3.1}$$

$$\textsc{LatentLoss} = -\frac{1}{2}(1 + \log(\det\boldsymbol{\Sigma}) - \operatorname{tr}(\boldsymbol{\Sigma}) - \boldsymbol{\mu}^T\boldsymbol{\mu}) \tag{3.2}$$

$$\textsc{TotalLoss} = \textsc{ReconstrLoss} + \textsc{LatentLoss} \tag{3.3}$$

For computational reasons, we denote the diagonal elements of $\boldsymbol{\Sigma}$ (those are the only non-zero elements) as a vector, and use the logarithm of the vector as the dense layer output.

A demo visualization of VAE is shown in Figure 3.1. This demo has three dense layers each as decoder and encoder.

3.1.3 Dimensionality reduction (vs. PCA)

How does VAE perform in reducing dimensions? To visualize the performance, we set the dimension of $\boldsymbol{z}$ to be 2. In this experiment, we set epoch to be 50. The training takes about 4 minutes. Figure 3.2 shows the scatter plot. It's the immediate fact from the plot that VAE seems to be more scattered.

[3]Technically, we cannot subtract a vector from a number. This means we do the operation on every dimension. This is consistent with the implementation using Tensorflow.

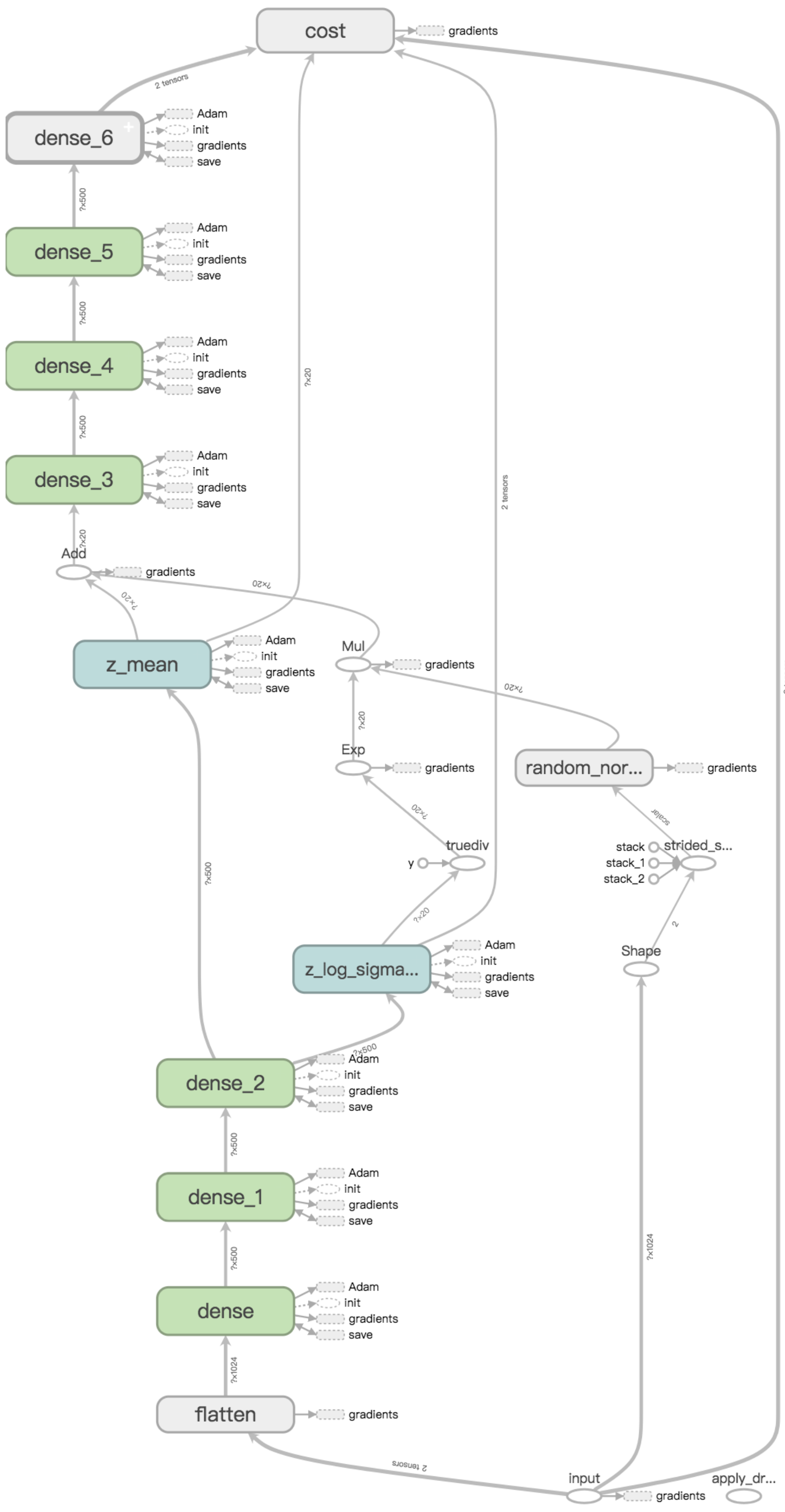


Figure 3.1: VAE diagram. Three layers each. Generated by Tensorboard

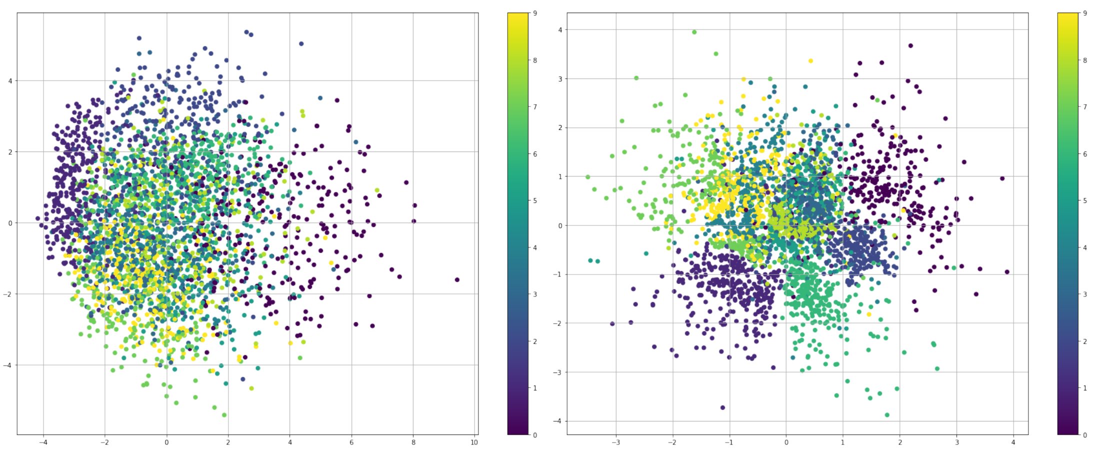

(a) Result with PCA (b) Result with VAE

Figure 3.2: Dimensionality reduction performance comparison

Here is something that PCA is unable to do: reconstruction. We iterate over $\boldsymbol{z} = [z_1, z_2]^T$ for $2.5 \leq z_i \leq 2.5$, and we get Figure 3.3[4]. The image looks very blurry, especially on the border. And some numbers like 5 and 9, takes very little space before it blends in with others. We will see that this actually brings difficulty to the classification task in the next section.

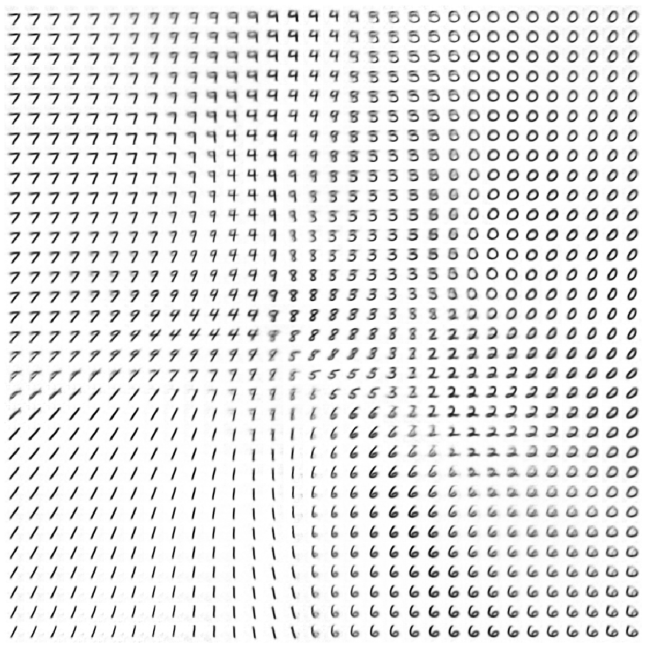


Figure 3.3: VAE 2D reconstruction

3.1.4 Work with CNN

Combining CNN with VAE, we use "conv" in the encoder layers, and "deconv" in the decoder layers, we can get the structure as shown in Figure 3.4.

How well is the performance of this network? We tried to apply some reconstruction on some randomly picked test data, as shown in Figure 3.5. We can see that this is even sharper than standard-dense-layered VAE. Also notice that the reconstruction network already make assumptions, for ex-

[4]This figure is inversed in color for better outlook on a white sheet

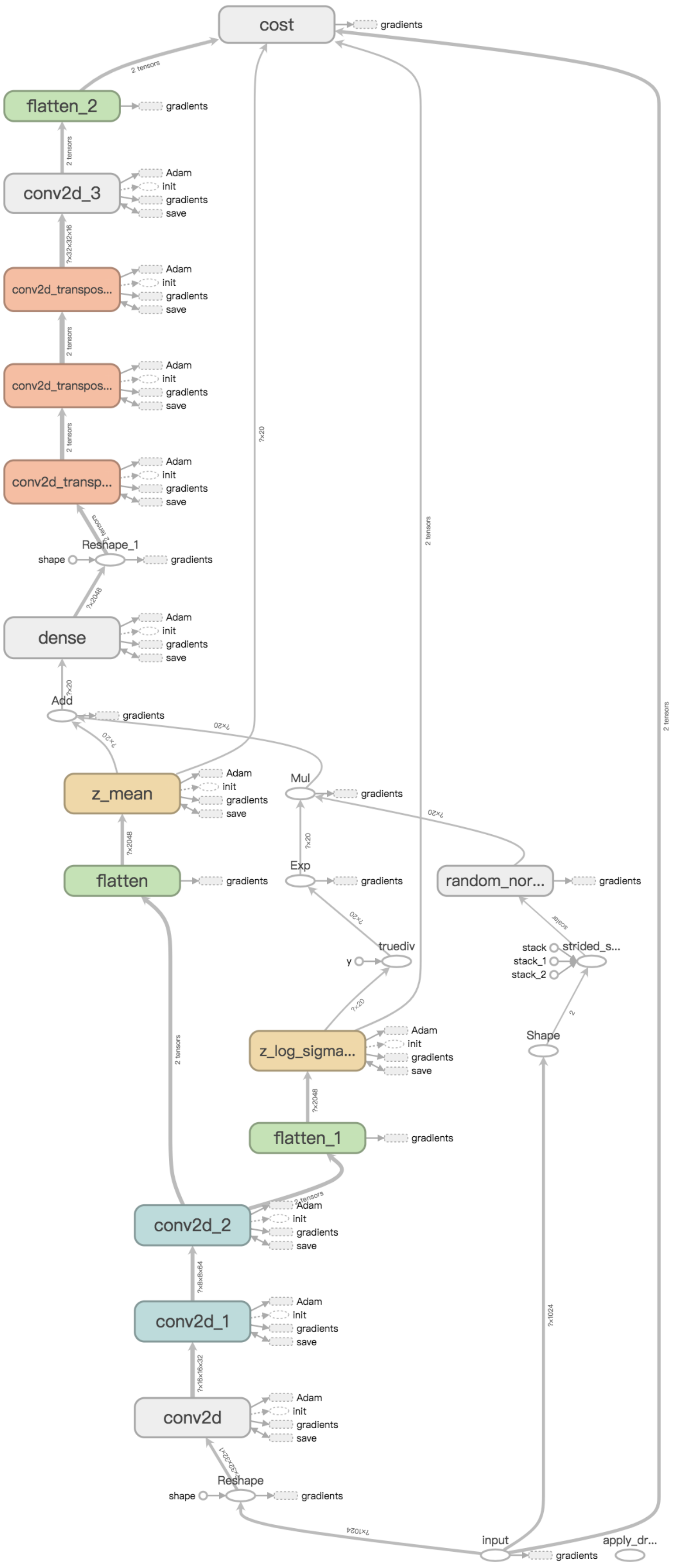


Figure 3.4: VAE with CNN. Generated by Tensorboard

ample, the 5 is believed to be 6. Although, currently the network knows nothing about what is 5 or what is 6. Also, compared to PCA, who is doing linear transformation as usual, VAE generally has less distortions.

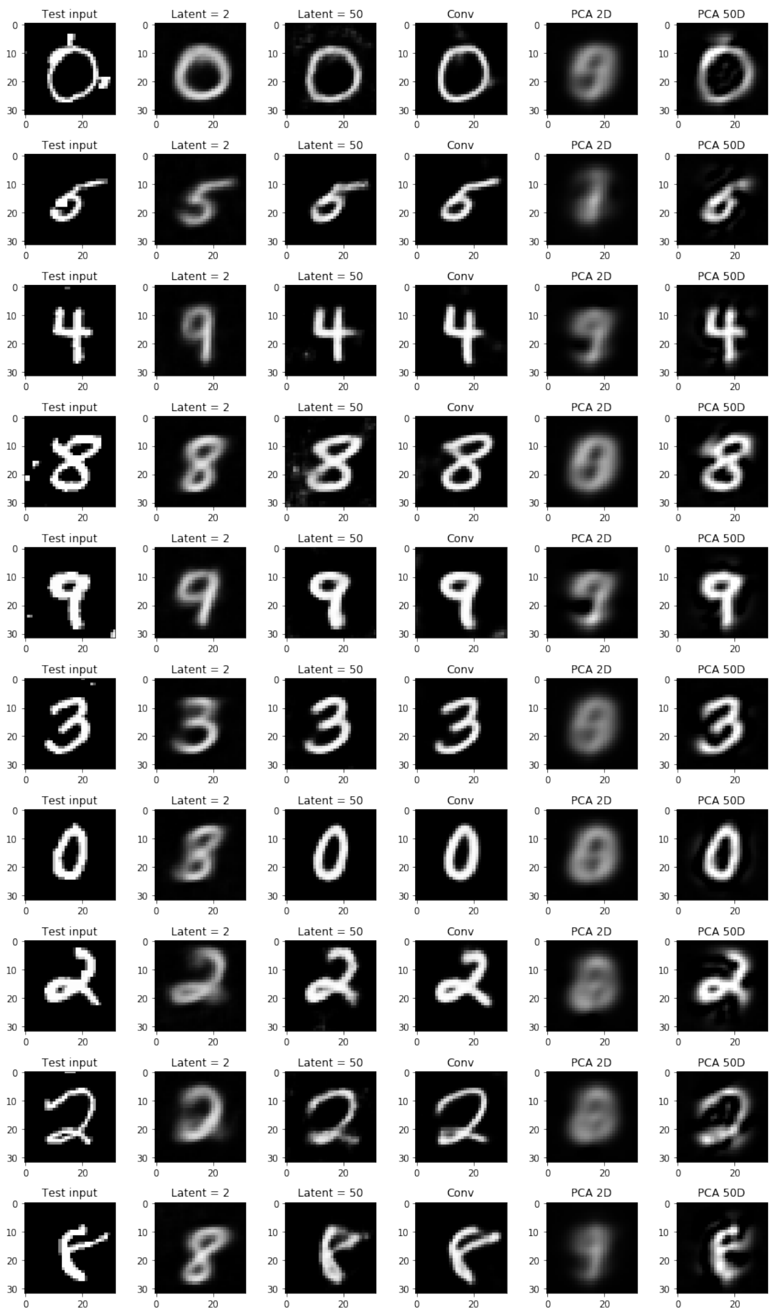


Figure 3.5: VAE / CNN / PCA. Comparison

3.1.5 Classification

It might be disappointing that when we actually use the latent variable z, with SVM, to do the classification, the performance does not even outcompetes the simple method using PCA + SVM. Even using a convolutional VAE that has been trained for about 8 hours, the accuracy is 0.9634 – still a little bit less than the model with PCA. We tried to improve this with the information provided in the confusion matrix, but hardly helpful.

$$\begin{bmatrix}
977 & 1 & 0 & 1 & 0 & 2 & 7 & 1 & 1 & 2 \\
0 & 1177 & 5 & 2 & 0 & 0 & 2 & 5 & 2 & 0 \\
1 & 3 & 1001 & 2 & 3 & 3 & 1 & 6 & 9 & 2 \\
0 & 0 & 14 & 977 & 1 & 12 & 3 & 3 & 15 & 5 \\
1 & 3 & 3 & 0 & 887 & 0 & 4 & 6 & 1 & 22 \\
2 & 1 & 1 & 16 & 0 & 849 & 12 & 0 & 10 & 3 \\
4 & 3 & 0 & 3 & 3 & 5 & 1001 & 0 & 1 & 1 \\
2 & 5 & 8 & 4 & 9 & 1 & 1 & 978 & 0 & 20 \\
2 & 2 & 4 & 12 & 2 & 8 & 3 & 4 & 875 & 10 \\
3 & 1 & 2 & 2 & 17 & 3 & 2 & 14 & 6 & 912
\end{bmatrix}$$

It's not intuitive why reconstruction of VAE seems to be look better but performs badly on the classification task. One possible reason is that the information machines need for classification is not exactly the information that is needed for reconstructing human-friendly images.

3.2 Conditional Variational Auto Encoder

Conditional VAE is not a new concept. In short, it modifies VAE by encoding $[\boldsymbol{x}, \boldsymbol{y}]$ instead of $\boldsymbol{x}$, and decoding $[\boldsymbol{z}, \boldsymbol{y}]$ instead of $\boldsymbol{z}$.

Again, out of the mysterious belief that $\boldsymbol{y}$ should not go into some uncontrollable network, we first added $\boldsymbol{z}$ at the last layer, that is, the layer to calculate the mean and variance of $\boldsymbol{z}$, instead of concatting the vectors before everything. Later we found that feeding $\boldsymbol{z}$ into every layer would possibly do better – this would, intuitively, increase the degree to which labels matter.

Due to lack of reference, building a CVAE is completely out of intuition and experimental feedbacks. We add some dropout layers to prevent overfitting. The whole network structure is shown as in Figure 3.6.

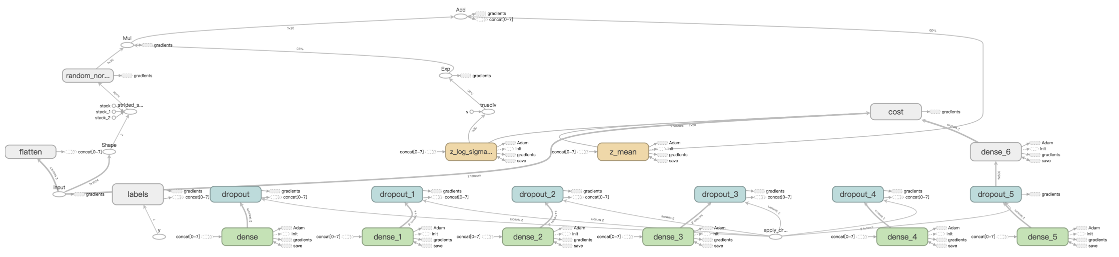


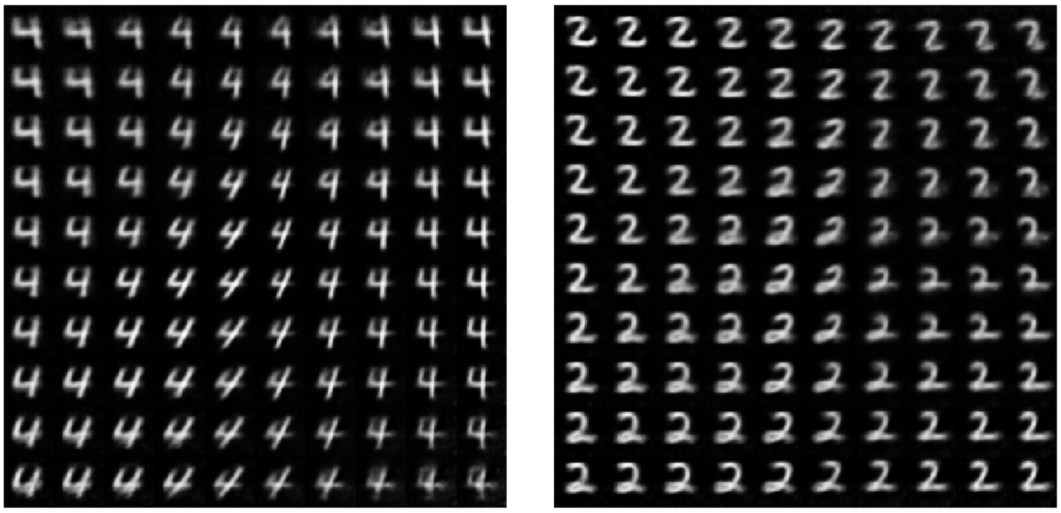
Figure 3.6: CVAE. Generated by Tensorboard

A representative application of CVAE is that we can control the label and generate different styles of a number, as shown in Figure 3.7.

How exactly does CVAE help in classification? Note that what CVAE is good at, is also reconstruction. The advantage is it can generate the answer given a label. Therefore we can guess a label and use the guessed label and raw image to reconstruct a new image. We calculate the reconstruction loss to determine which guessed label will give us the minimum loss. The algorithm pseudo code is demonstrated in Algorithm 1.

Figure 3.8 shows the reconstruction of the image using labels from 0 to 9.

The reconstruction effect seems reasonably okay, it also helps **denoising**: you can see almost all the noises in the input image has been magically removed; but the accuracy is relatively low (less than 95%). To investigate the reason, we tried to plot the reconstruction details of the images that are predicted incorrectly. This is Figure 3.9.



(a) Different styles of 4 (b) Different styles of 2

Figure 3.7: CVAE generates different styles of numbers

Algorithm 1 PREDICT(x)

```
Require: $\boldsymbol{x}$ is a $n$-dimension vector.
  for all $y$ in $[0, 9]$ and $y$ is an integer do
    regenerate $\boldsymbol{x}'$ with $(\boldsymbol{x}, y)$.
    BestLoss $\leftarrow \infty$
    if ReconstrLoss$(\boldsymbol{x}, \boldsymbol{x}')$ $<$ BestLoss then
      BestLoss $\leftarrow$ ReconstrLoss$(\boldsymbol{x}, \boldsymbol{x}')$
      Label $\leftarrow y$
    end if
  end for
  return Label
```

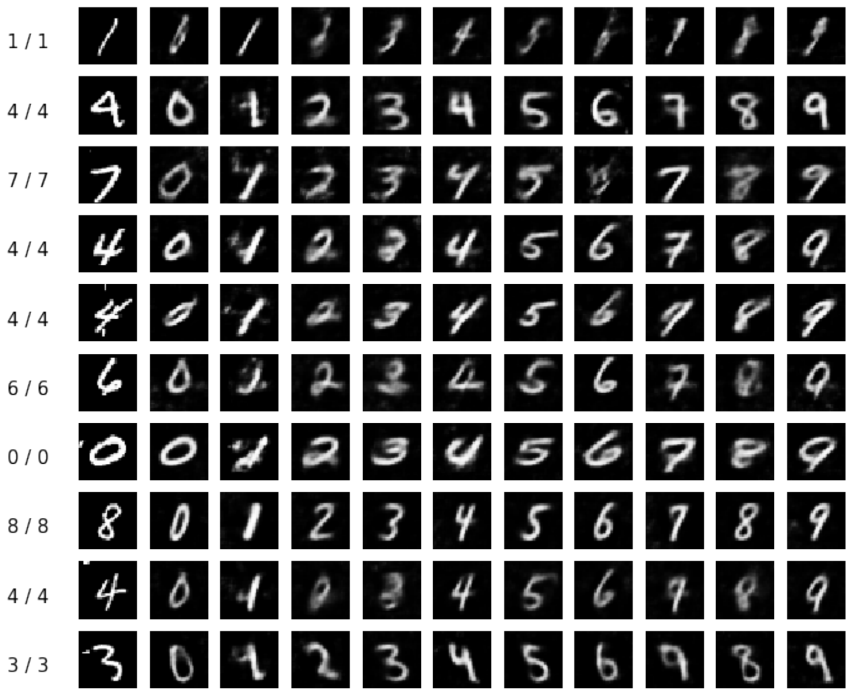


Figure 3.8: CVAE reconstruction with label from 0 to 9

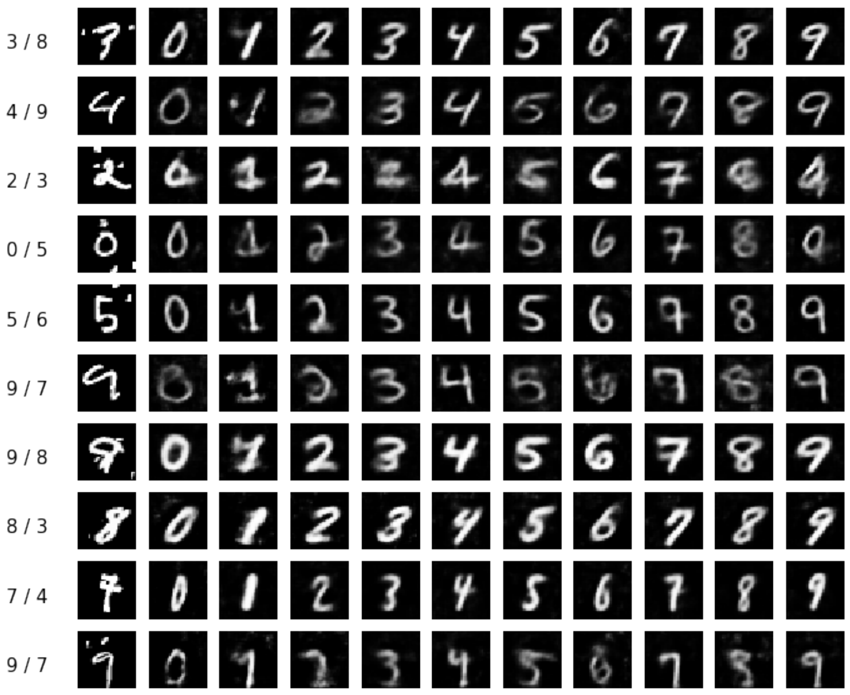


Figure 3.9: Errors from the first 200 samples

It seems that the image is reconstructed "incorrectly" because they are adjusted to average shape during the reconstruction, which results in the deviation from the raw image data. The key idea why this algorithm should work is the belief that when using the correct label, the input image will not change much. As a matter of fact, these images did not change much in the shape, but in terms of the loss function, someone else, with a different shape, outcompetes them.

Now we had no clue about how to solve this problem, and had to give up.

NOTE We suffer strange issue that the cost will be NaN after dozens of epochs. We didn't solve the problem, but used the checkpoint to restore the latest useful version.

4 Benchmark

The benchmark test is done on a server with Intel Xeon CPU E5-2650 v4 @ 2.20GHz with 24 cores and 32 GB RAM, without GPU.

The time is not actually the CPU time usage, but the real time usage, because parallelism of the program is also an important metrics. We don't consider it a burden if a program uses more cores than another one. Also notice that the time includes training time and evaluation time. (Intuitively, kNN is slow because it has long evaluation time.)

The benchmark is grouped into 3 parts, which correspond to: traditional methods, CNN and VAE. In the "traditional" group, SVM + PCA is the best method. We get approximately 96.8%, which is better than all the attempts with VAE. CNN gives us the global best, about 99.1%. It might be surprising that the autoencoder group performs poorly in classification. However, autoencoders might be able to give us more knowledge than simply classify them. We can even denoise the images using CVAE.

The full benchmark is shown in Table 3.

Method	Preprocess	Accuracy	Time (second)	Comment
MLP	None	0.6027	296.6	iter = 101 before converge
MLP	Centering	0.8908	79.0	iter = 48 before converge
MLP + PCA [5]	None	0.8372	100.6	iter = 165
MLP + PCA	Centering	0.9432	112.7	iter = 198
kNN	Thresholding	0.7057	368.5	eval time is 332.5
kNN	Centering	0.9306	196.9	eval time is 160.8
kNN	Thres + Cent	0.9011	161.3	eval time is 124.4
SVM + PCA	Thresholding	0.882	353.3	
SVM + PCA	Centering	**0.9678**	144.1	
SVM	Thres + Cent	?	∞ [6]	
SVM + PCA	Thres + Cent	0.9641	147.4	
CNN (TF Demo)	Centering	0.9481	3606.9	step = 20000
CNN	Centering	0.9723	57.9	epoch = 1
CNN	Centering	0.9832	178.8	epoch = 3
CNN	Centering	**0.9905**	1219.5	epoch = 200
VAE + SVM	Centering	0.6944	$\approx$ 400	latent dim = 2
VAE + SVM	Centering	0.9403	$\approx$ 600	latent dim = 50
VAE (Conv) + SVM	Centering	0.9634	$\approx$ 8 hours	
CVAE	Centering	0.9423	1297.5	

Table 3: Benchmark result

5 Conclusion

This section is intentionally left blank.

6 Code Repo

The code repository is available at `https://github.com/ultmaster/MNIST-Toy-Machine`.

Since we have too many unsuccessful attempts. all the "ridiculous" models, even those that have possibly been mentioned in this report due to their nonsense, have been moved to the `old` directory. There is also a notebook directory, containing the drafts, visualization code, and some other surprises. (It contains a lot of chaos; you may not want to check that.)

[6]We failed to wait for the result.