Chen Gong
518030910301
gongchen@sjtu.edu.cn

Report for Project 1
CS385 Prof.Quanshi Zhang

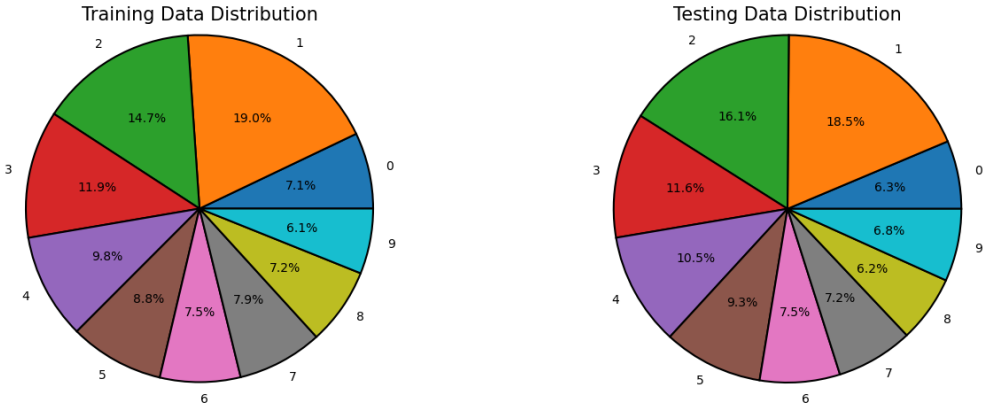
2021-06-18

1 Data Process

In the experiment, I use SVHN of handwritten digits as dataset which includes over 70000 training data samples and 20000 testing data samples. For each data sample(picture), if I directly crop the digit number using its given bounding box, resizing it to 32×32 leads to distortions. Therefore, I extend the original boxes in the appropriate dimension to square windows.
Due to the dataset's large size, I reduce the size to $\frac{1}{7}$ original size by random choosing $10000+$ pictures as training data and $3500+$ pictures as testing data. Then:

- for logistic regression-based models, LDA and SVM, I extract the HOG (Histogram of Oriented Gradient) features of each picture which is recommended by teacher in class to improve the performance of model. In this way, each data sample's dimension is transformed from $3 \times 32 \times 32$ to 324.
- for Neural Networks and generative models, I directly use the normalized cropped pictures as training and testing data.

The distributions of training data and testing data are shown in figure 1.



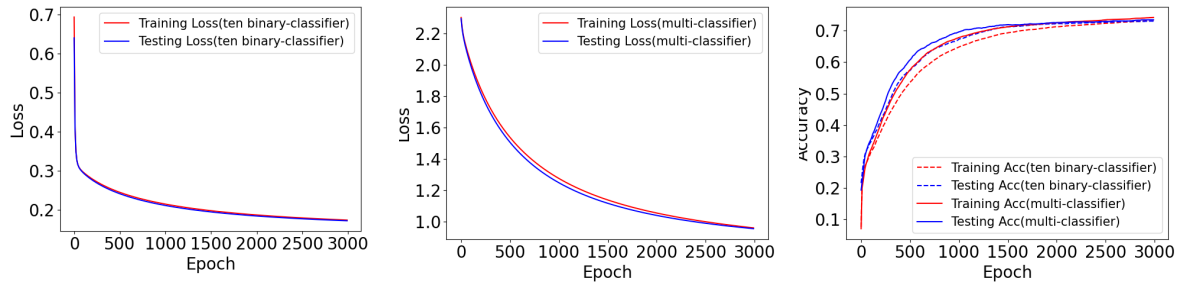
(a) Training Data Distribution

(b) Training Data Distribution

Abbildung 1: Distributions of training data and testing data

2 Logistic Regression

In this section, I will show the experimental results and analysis of original LG(logistic regression) model, LG with Ridge loss, LG with Lasso loss, kernel-based LG with Lasso loss. To simplify explanation, we use $beta$ to denote model parameter, $L(\beta)$ to denote corresponding loss function, $X = \{X_1, \ldots, X_n\}$ to denote data, $Y^* = \{y_1^*, \ldots, y_n^*\}$ to denote true labels. We use $\beta_k, k \in K$ to denote the parameter of binary-classifier for class k.
Basically, LG model can only fit binary classification task using log likelihood as loss function and $sigmoid()$ as output processing function, which means that multi-classification can only be achieved by constructing multiple binary classifiers. After searching for related work, I learn that LG can be developed to multi-classification task by revising the dimension of parameter β, using cross entropy as loss function and $softmax()$ as output processing function. To find their differences, for all the LG-based models, I implement both 10 binary-classifiers and direct multi-classifiers and compare their performance.
Note that for ten binary-classifiers, I use 'OvR'(One versus Rest) principle to transform the training labels, and the prediction of model is the class with largest output $y_i = X_i^\top \beta_k$ where $k \in K$ denotes the classes.



(a) Average Loss of ten binary-classifiers (b) Loss of multi-classifier (c) Accuracy

Abbildung 2: Basic logistic regression results. Red lines denote training results and blue lines denote testing results.

2.1 Basic Logistic Regression

2.1.1 Principles

In logistic regression ,we use log likelihood(binary-classifier) and cross-entropy (multi-classifier) as loss function $L(\beta)$:

$$\begin{aligned} L(\beta) &= -\frac{1}{n}\sum_{i=1}^{n}[y_i^* X_i^\top \beta - \log(1+\exp X_i^\top \beta)] \\ L(\beta) &= -\frac{1}{n}\sum_{i=1}^{n}[y_i^* \log(\text{softmax}(X_i^\top \beta))] \end{aligned} \tag{1}$$

The updating rules for two methods are:

$$\begin{aligned} \beta &\leftarrow \beta - \frac{1}{n}\sum_{i=1}^{n}[(\text{sigmoid}(X_i^\top \beta) - y_i^*)X_i] \\ \beta &\leftarrow \beta - \frac{1}{n}\sum_{i=1}^{n}[X_i^\top(\text{softmax}(X_i^\top \beta - y_i^*))] \end{aligned} \tag{2}$$

2.1.2 Hypotheses, Verification and Analysis

1. Hypothesis: It is easier to train a binary-classifier than a multi-classifier and thus the binary-classifier is expected to converge more quickly and

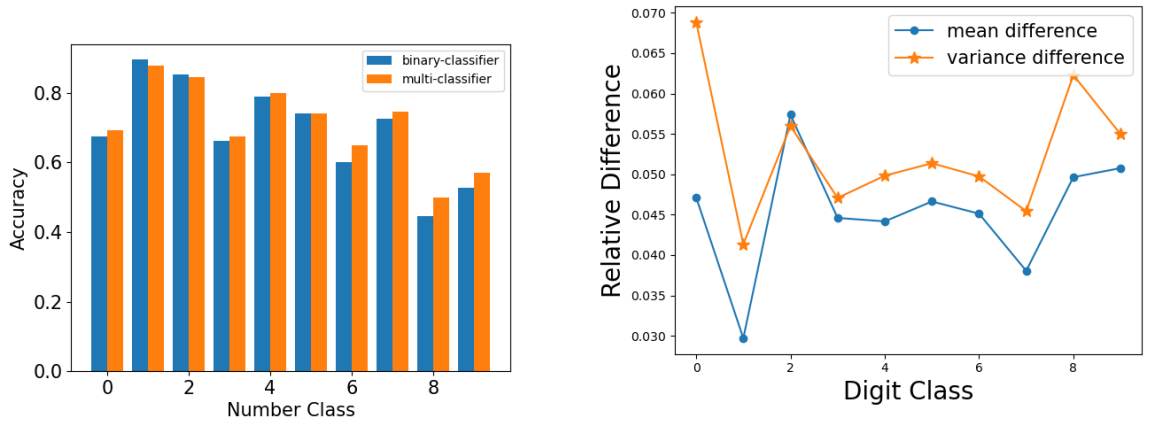
   Verification: Comparing results shown in figure 2(a) and 2(b), we can directly see that the average loss of binary-classifiers decreases more quickly and sharply in first 30 epochs, which demonstrates our hypothesis. It seems that binary-classifier has better performance in real situations as it needs less training epochs to converge. However, we also need to consider the time complexity as ten binary-classifiers' training time for one epoch is theoretically ten times the multi-classifier's training time.

2. Hypothesis: ten binary-classifiers has better performance on judging (whether the digit in picture belongs to the class we gave) but worse performance on detecting (what exact class the digit number belongs to).

   Verification: Comparing binary-classifier's and multi-classifier's judging abilities is meaningless and unfair in real situation. Thus, we only compare their detecting ability. The two ways' accuracy of different digit numbers classes can be seen through figure 3(a). We can see that for most classes, multi-classifier's detecting ability is stronger which coincides our expectation. Only on two classes (number 1 and 2), ten binary-classifiers have better performance. Through analyzing, I find that this is because the data sizes of these two classes are bigger shown in figure 1, which means that when given sufficient training data, multiple binary-classifiers can achieve the same detecting ability as multi-classifier.

3. It is strange to see that the testing accuracy is a little bit higher than the training accuracy in the first 2000 epochs. In my opinion, the reason is that in the beginning LG model cannot learn the characteristics of training data totally and there exists little difference between the distributions of training data and testing data.

   To verify this point, I plot each digit class's difference of mean μ and variance σ value between training data and testing data. To illustrate more clearly, I show the ratio of difference to the absolute mean value of training data. As is shown in 3(b), the largest difference of mean is only 6% and the largest difference



(a) Accuracy for different classes

(b) Difference of Data Hog Feature mean and variance.

Abbildung 3: The above two figures shown the accuracy of different classes and the distribution difference of different classes.

of variance is only 7% which demonstrates the little difference between training data's and testing data's distributions of each digit class.

2.2 Ridge Loss

2.2.1 Principle

When adding ridge loss, the loss function for ten binary-classifiers and multi-classifier become:

$$
\begin{aligned}
L(\beta) &= -\frac{1}{n}\sum_{i=1}^{n}[y_i^* X_i^\top \beta - \log(1+\exp X_i^\top \beta)] + \frac{1}{2}\lambda \parallel \beta \parallel^2 \\
L(\beta) &= -\frac{1}{n}\sum_{i=1}^{n}[y_i^* \log(\mathrm{softmax}(X_i^\top \beta))] + \frac{1}{2} \parallel \lambda \parallel^2
\end{aligned}
\tag{3}
$$

The updating rules for two methods become:

$$
\begin{aligned}
\beta &\leftarrow \beta - \frac{1}{n}\sum_{i=1}^{n}[(\mathrm{sigmoid}(X_i^\top \beta) - y_i^*)X_i] - \lambda\beta \\
\beta &\leftarrow \beta - \frac{1}{n}\sum_{i=1}^{n}[X_i^\top(\mathrm{softmax}(X_i^\top \beta - y_i^*))] - \lambda\beta
\end{aligned}
\tag{4}
$$

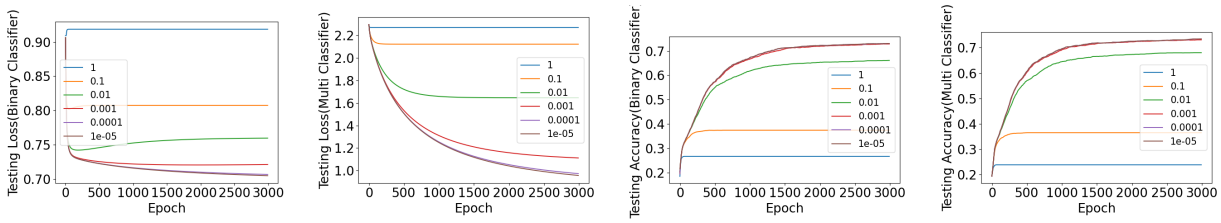
2.2.2 Hypotheses, Verification and Analysis

1. Hypothesis: Theoretically, adding regularization term with suitable weight can improve model's generalization ability and robustness by avoiding over-fitting. Thus, we can find the most appropriate regularization term λ by conducting experiments with λ ranging from 10^{-5} to 1.

   Verification: Empirical results shown in figure 4 demonstrate that the larger λ is, the worse the model performance is (lower testing accuracy and higher testing loss), which contradicts our hypothesis that the addition of ridge loss can improve the generalization ability of model. This result is quite amazing beyond our knowledge. The reason is that the difference between training data and testing data is so little that it can be ignored, as is shown in figure 3(b). Therefore, in stead of making the model more generalized, the regularization term even prevents the model from reaching the best performance state, which bring about worse performance.

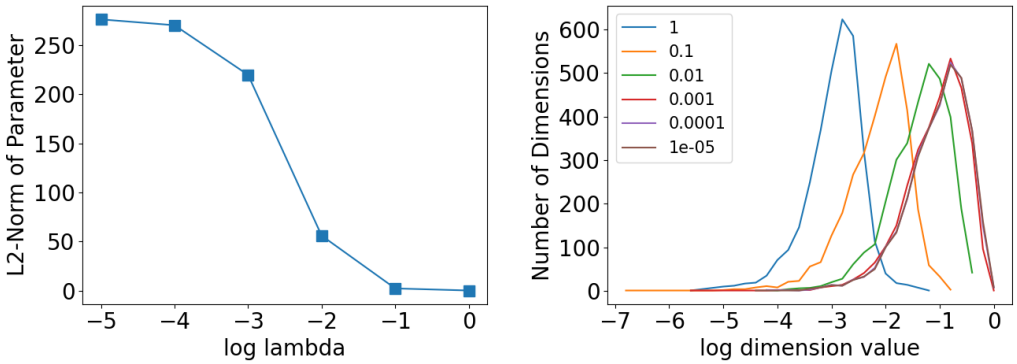
2. Hypothesis: The regularization term $\frac{1}{2}\lambda \parallel \beta \parallel^2$ penalizes the weight dimension dimensions with large absolute values and prevents the regression from being conducted based on a few feature dimensions.

   Verification: We plot the distribution of parameter β and its l2-norm. Figure 5(b) shows the distribution of dimension values with different λ, which demonstrates that λ reduces the number of dimensions with large absolute values by moving the whole distribution towards smaller direction.



(a) Testing Loss of Binary-Classifier (b) Testing Loss of Multi-Classifier (c) Testing Accuracy of Binary-Classifier (d) Testing Accuracy of Multi-Classifier

Abbildung 4: Ridge Loss Results with Different regularization weight λ.



(a) Ridge loss: l2-norm of model parameter (b) Ridge loss: Distribution of model parameter dimension values

Abbildung 5: The parameter information when adding different regularization weight λ. The left figure shows the square of l2-norm, i.e. $\| \beta \|^2$. The right figure shows the distribution of the elements of parameter β. Note that we only show the parameter of multt-classifier which has 324×10 dimensions in total.

3. Analysis: From figure 5(b) we can also see that only a few dimensions are relatively small or relatively huge which proves the saying in books: model uses a large number of feature dimensions for regression and each dimension contributes a little to the regression result.
4. Analysis: From figure 5(a), we can directly see the huge impact of λ on final parameter.

2.3 Lasso Loss

2.3.1 Principle

When adding Lasso loss, the loss function for ten binary-classifiers and multi-classifier become:

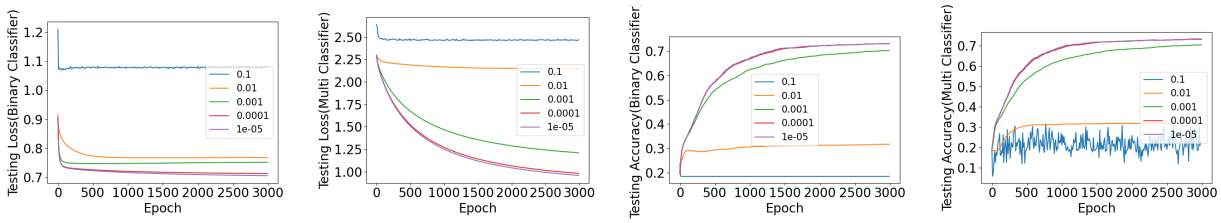
$$\begin{aligned} L(\beta) &= -\frac{1}{n}\sum_{i=1}^{n}[y_i^* X_i^\top \beta - \log(1+\exp X_i^\top \beta)] + \lambda \parallel \beta \parallel_{l_1} \\ L(\beta) &= -\frac{1}{n}\sum_{i=1}^{n}[y_i^* \log(\text{softmax}(X_i^\top \beta))] + \lambda \parallel \beta \parallel_{l_1} \end{aligned} \tag{5}$$

The updating rules for two methods become:

$$\begin{aligned} \beta &\leftarrow \beta - \frac{1}{n}\sum_{i=1}^{n}[(\text{sigmoid}(X_i^\top \beta) - y_i^*)X_i] - \lambda \text{sign}(\beta) \\ \beta &\leftarrow \beta - \frac{1}{n}\sum_{i=1}^{n}[X_i^\top(\text{softmax}(X_i^\top \beta - y_i^*))] - \lambda \text{sign}(\beta) \end{aligned} \tag{6}$$

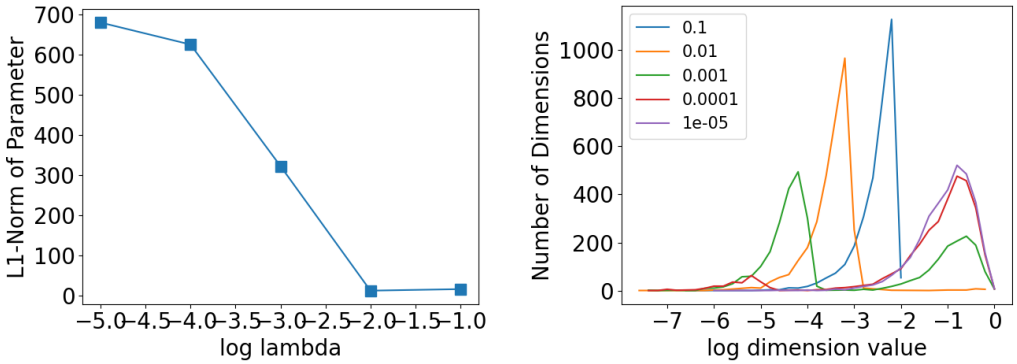
2.3.2 Hypotheses, Verification and Analysis

1. Hypothesis: Just like ridge loss, we think that adding lasso loss term with suitable weight λ can improve model's generalization ability and robustness by avoiding over-fitting. Through tuning hyper-parameter λ from 10^{-1} to 10^{-5}, we can find the most appropriate regularization term λ.



(a) Testing Loss of Binary-Classifier (b) Testing Loss of Multi-Classifier (c) Testing Accuracy of Binary-Classifier (d) Testing Accuracy of Multi-Classifier

Abbildung 6: Lasso Loss Results with Different regularization weight λ.



(a) Lasso loss: l2-norm of model parameter (b) Lasso loss: Distribution of model parameter dimension values

Abbildung 7: The parameter information when adding different regularization weight λ. The left figure shows the square of l2-norm, i.e. $\| \beta \|^2$. The right figure shows the distribution of the elements of parameter β. Note that we only show the parameter of multt-classifier which has 324×10 dimensions in total.

Verification: Empirical results shown in figure 6 contradicts our hypothesis. The larger λ is, the worse model performance is. This result is similar to the situation of ridge loss except that the influence of lasso loss seems to be less than ridge loss. This is also reasonable because as lasso loss only penalizes with 1 or -1 while ridge loss penalizes more, the same degree as β.

2. Hypothesis: adding term of lasso loss stands for "least absolute shrinkage and selection operator"which means that only a small number of components of β are zero.

   Verification: Just like ridge regression, I also show the distribution of model parameter dimension values with different weight λ. The result is strange. When $\lambda = 0.001$, the model parameter dimension value is centered at $10^{-4.5}$ and 10^{-1}. Then, when λ becomes either bigger or smaller, the distribution of dimension values is moving towards the increasing direction. I try to search for more papers to explain this strange phenomenon but time is not enough for me to totally find the reason. In this summer, I will re-experiment this part to see whether this phenomenon is caused by my coding mistake and try to discover more things from it. Many thanks!

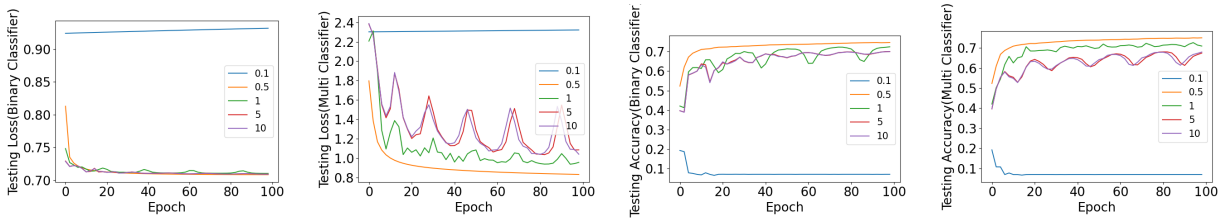
3. Analysis: From figure 6, we can also see that when λ is large enough like 0.1, the model will vibrate violently as the testing loss and accuracy are fluctuating. This is because that when the model converges, many dimensions become near to zero. Every time we use the gradient including $sign()$ function to update model parameter, the parameter will fluctuate around 0 from negative to positive or from positive to negative because of the suitable weight λ 0.1. Empirical results also demonstrate my explanation.

2.4 Kernel-Based

2.4.1 Principle

We sometimes use $\phi(x)$ instead of x as features to deal with the regression problem. The output $f(x)$ can be expressed as:

$$f(x) = \sum_{i=1}^{n} c_i K(x, x_i) = \sum_{i=1}^{n} c_i < \phi(x), \phi(x_i) > = < \phi(x), \sum_{i=1}^{n} \phi(x_i) > \tag{7}$$



(a) Testing Loss of Binary Classifier (b) Testing Loss of Multi Classifier (c) Testing Accuracy of Binary Classifier (d) Testing Accuracy of Multi Classifier

Abbildung 8: The testing loss and testing accuracy of ten binary-classifiers and multi-classifier using rbf kernel with different parameters σ.

where we define $K(x, x_i) := \phi(x)^\top \phi(x_i)$. Normally, we use four types of kernel:

- rbf: $K(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$
- polynomial: $K(x_i, x_j) = (x_i^\top x_j)^d$
- cos: $K(x_i, x_j) = \frac{x_i^\top x_j}{\|x_i\|\|x_j\|}$
- sigmoid: $K(x_i, x_j) = \tanh(\alpha x_i^\top x_j + c), \tanh(\alpha) = \frac{1-\exp(-2\alpha)}{1+\exp(-2\alpha)}$

In my opinion. kernel function essentially change each data sample's features from original self-feature into new features which consists of the correlation and relation with other data samples. In this way, the prediction of one data sample is decided by its relationship(like distance) with other data samples from various classes and the model actually learns how to combine these relationships to obtain the correct prediction.

2.4.2 Hypothesis, Verification and Analysis

1. Hypothesis: for rbf kernel, we want to see the influence of hyper-parameter σ. Theoretically, the larger σ is, the less $K(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$ is and the closer it is to 1. Also, the difference between different pair $K(x_i, x_j)$ becomes smaller.

   Verification: We conduct experiments with σ ranging from 0.1 to 10 to see the change of model performance. The experimental results can be seen in figure 8. Apart from the situation when $sigma = 0.1$, the model performances with different σ values coincide our expectation. The smaller σ is, the difference between different pairs $K(x_i, x_j)$ becomes bigger, which means that different data sample's features are more distinguishable and thus the model can learn them more easily with better performance. However, when σ is small enough such as 0.1, even two data samples from the same digit class can have really different features which means that data samples become more dispersed and more difference. Therefore, model cannot successfully learn the feature of each class and the loss will never decrease.

2. Phenomenon: From 8, we see that the testing loss and accuracy of rbf kernel will fluctuate periodically like the results of lasso loss shown in figure 6. And the larger σ is, the fluctuation is more violently.

   Hypothesis: I think that the reason is the same as what I said in lasso loss part. Most of the parameter dimension values are near to the lasso loss weight λ we set which is 10^{-3} and thus every time the parameter is updating, it will vibrate around 0.

   Verification: I plot the distribution of parameter dimension values, which is shown in figure 9(a). We can see that when $\sigma = 1, 5, 10$, most of the parameter dimension values are centered at around 10^{-3} which is just the value of lasso loss weight λ. Thus, my explanation is correct.

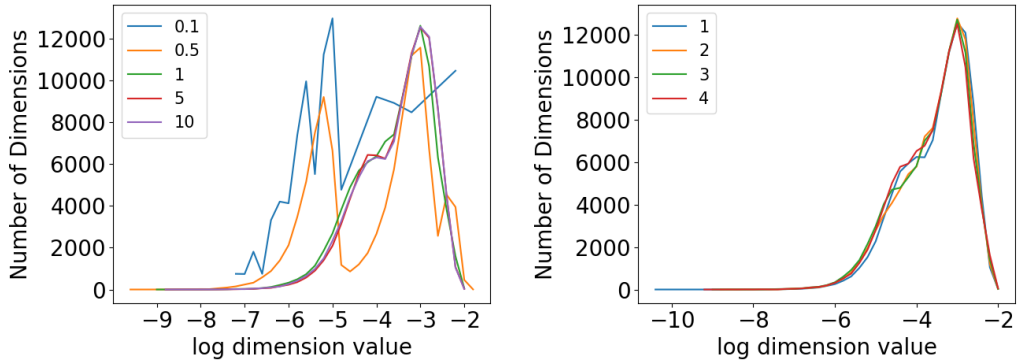
3. Hypothesis: for the polynomial kernel, the larger d is, the more complex $K(x_i, x_j) = (x_i^\top x_j)^d$ is, the bigger kernel space is and it is more likely to separate data samples from different classes.

   Verification: Empirical results shown in figure 10 demonstrate our hypothesis that larger d leads to lower testing loss and higher testing accuracy.

4. Phenomenon: As is shown in figure 10, all the model performances of polynomial kernel will fluctuate periodically.

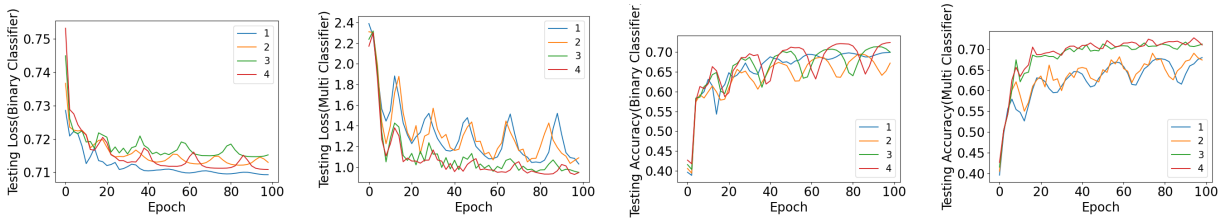
   Analysis: The reason is the same as that in rbf kernel: most of the parameter values are near to lasso loss weight λ which is 10^{-3} and thus every time parameter is updated, it will vibrate around 0.

   Verification: We plot the distribution of parameter dimension values in figure 9(b). It is clear to see that all the models' parameter dimension values are centered around 10^{-3} which verifies our analysis.



(a) rbf (b) poly

Abbildung 9: Distribution of model parameter dimension values



(a) Testing Loss of Binary Classifier (b) Testing Loss of Multi Classifier (c) Testing Accuracy of Binary Classifier (d) Testing Accuracy of Multi Classifier

Abbildung 10: The testing loss and testing accuracy of ten binary-classifiers and multi-classifier using polynomial kernel with different parameters d.

5. Phenomenon: As is shown in figure 12, the best model performance is achieved when $\alpha = 1$.

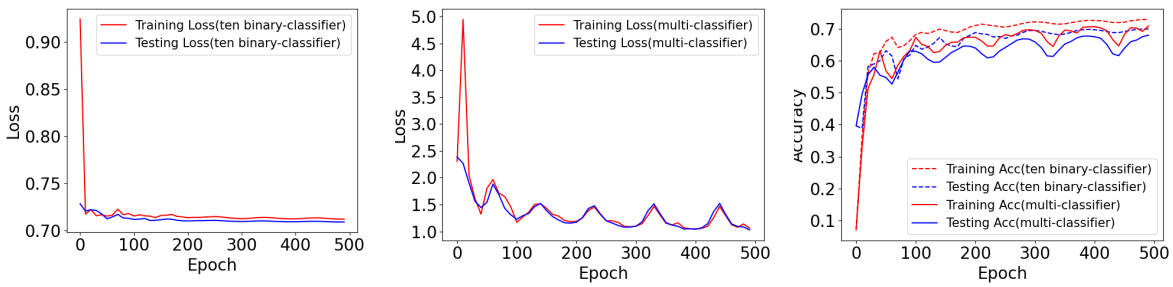
   Analysis: This is a strange phenomenon which deserves deep thinking and experimenting. I am so sorry that my time left is not enough and I will continue to discover the explanation for this in summer vacation.

We also conduct the experiments on cos kernel which is shown in figure 11. It also fluctuates periodically which has already been explained before.

3 LDA

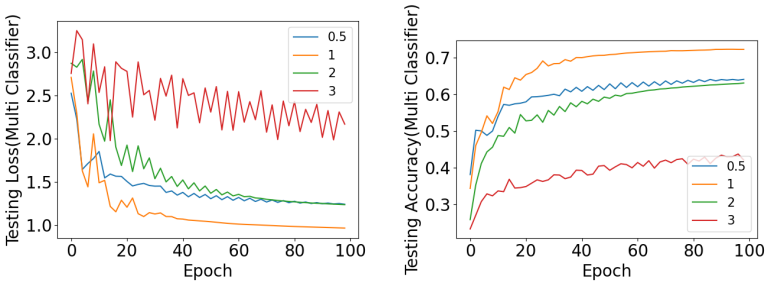
3.1 Principles

There are two forms of LDA, one for binary-classification and one for multi-classification task.
Essentially, LDA is a method to reduce the dimension and then do classification task. In order to do dimension reduction, we need to compute β which maximizes inter-class variance S_w and minimizes the intra-variance S_b.



(a) Loss of Binary-Classifier (b) Loss of Multi-Classifier (c) Accuracy

Abbildung 11: Cos kernel: Loss and Accuracy of ten binary-classifiers and multi-classifier



(a) Testing Loss (b) Testing Accuracy

Abbildung 12: Sigmoid kernel: Testing loss and accuracy of multi-classifier

The computation process for binary-classification task is:

$$\begin{aligned} S_w &= \sum_{x\in\Omega_+}(x-\mu_+)(x-\mu_+)^\top + \sum_{x\in\Omega_-}(x-\mu_-)(x-\mu_-)^\top \\ S_b &= (\mu_+-\mu_-)(\mu_+-\mu_-)^\top \\ \Rightarrow \lambda\beta &= S_w^{-1}S_b\beta \end{aligned} \tag{8}$$

The computation for multi-classification task is:

$$\begin{aligned} S_t = S_w + S_b &= \sum_{i=1}^{n}(x_i-\mu)(x_i-\mu)^t op \\ S_w = S_t - S_w &= \sum_{i=1}^{n} n_i(\mu_i-\mu)(\mu_i-\mu)^\top \\ \Rightarrow \lambda\beta &= S_w^{-1}S_b\beta \end{aligned} \tag{9}$$

3.2 Experimental Details

The training and testing processes for two kinds of LDA are different.
In the training process for binary-classification LDA, I use ÖvR"(One versus Rest) method to create the training data for each class $k \in \{1, 2, \ldots, 10\}$. The whole process can be seen as follows:

1. Training process: For each class k, compute the eigenvectors and eigenvalues of $S_w^{-1}S_b$ when given class k's training data X_k. Choose d eigenvetors with largest d eigenvalues to compose β_k which is used to project data into a new space $\mathbb{R}^d$. Then, for each class k, we project its training data into the new k-dimensional space and compute its mean value μ_k as center of class.

3. Testing process: For each data sample x_i and each class k, we use matrix β_k to project it into d-dimensional space and compute its distance to the center μ_k. Finally, we regard the class k with least distance as prediction class.

The training process for multi-classification LDA is easier.

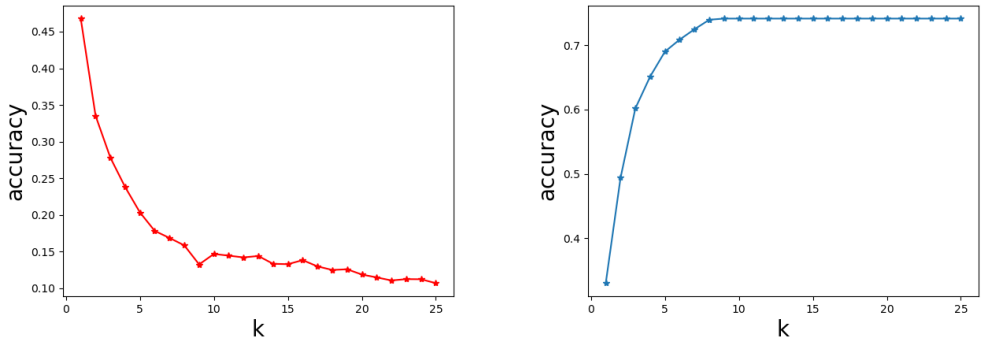
1. Training process: Instead of computing a matrix β_k for each class k, we directly compute one matrix β by computing the eigenvectors of $S_w^{-1}S_b$ and choosing d vectors with largest d eigenvalues to compose β. Then, for each class k, we input the training data X_k and obtain its center $\mu_k \in \mathbb{R}^d$.

2. Testing process: For each testing sample x_i, we use matrix β to project it into space with dimension d. Then, we find the closest center μ_k and regard class k as our prediction.

3.3 Results Analysis and Deep Thinking

1. Phenomenon: The results of applying binary-classification LDA is shown in figure 13(a). As is shown by the picture, the model performance is really bad, compared with multi-classification LDA shown in figure 13(b).

   Analysis: The reasons may be that: a) it is unfair and unreasonable to use the distance to class center μ_k to decide prediction class. We can use probability of $N(x_i|\mu_k, \sigma_k)$ instead. b) The binary-classification LDA is only suitable for judging task and works bad on classification task.



(a) Accuracy against dimension number k for binary-classification LAD

(b) Accuracy against dimension number k for multi-classification LDA

Abbildung 13: The testing accuracy for binary-classification LDA and multi-classification LDA

Kernel	Training Accuracy	Testing Accuracy
Linear	0.8218	0.7759
Poly	0.9820	0.8004
Rbf	0.8924	0.7977
Sigmoid	0.6830	0.6861

Tabelle 1: Model Performance of Different Kernel

2. Phenomenon: The testing accuracy of multi-classification LDA is increasing to over 70% when $k \in [1, 9]$ and then converges.

   Analysis: Essentially, LDA reduces the dimension of data. The larger number of dimension d we choose, the larger space data is projected to and the more information and features we extract. However, larger d means larger space to store β and more time to project the data sample through $x_i\beta$.

4 Support Vector Machines

4.1 Principle

Support vector machine essentially is a linear classification. The basic idea of it is to estimate a hyperplane that can separate two classes of samples. Because the mechanism and principles of are well-known to us. I do not introduce it here.

4.2 Empirical Results and Analysis

In the experimental result, I try four kinds of kernels: linear, polynomial, rbf and sigmoid.
Similar to previous sections, I randomly choose $10000+$ training data samples to train the SVM and obtain the hyper-planes used to do classification. Then, I randomly choose another $3500+$ testing data samples to test the performance of model.
The results can be seen as follows: From the table, we can see that on SVHN dataset, the model performances are $Polynomial > Rbf > Linear > Sigmoid$.

4.3 Analysis and Comparison

In this section, firstly I want to show my deep thoughts about the differences between SVM and LG(Logistic Regression):

1. The loss functions are different. LG uses log likelihood as loss function which reflects the probability and SVM uses the margin-based loss function which reflects the distance.

2. The hyper-planes computed by SVM is decided by the data samples near these hyper-planes. The model parameter computed by LG is decided by all the data samples.

Under this task, SVM clearly has better performance compared with LG. The reason is that the data of SVHN has a lot of noises. Although there exists bounding boxes helping to crop data, we still introduce noises of

numbers nearby if we crop the square area to avoid distortion. Thus, computing the probability like likelihood of data is not accurate. LG is more suitable to data with less noises.

5 Gaussian mixture methods

5.1 Principle

The GMM(Guassian Mixture Model) supposes that the total data distribution is consists of several Gaussian distributions. Each data x will fall within a cluster and its distribution is affected by its cluster:$p(x|z=k,\theta) \sim N(\mu_k, \sigma^2 I)$. Therefore, the final goal is to find the variables weight π_k, mean μ_k and variance Σ_k for each class k which could maximize the joint probability of all data samples.
We can use EM algorithm to iteratively:

1. E step: According to current π_k, μ_k, Σ_k to compute posterior probability $\gamma(z_{nk})$:

$$\gamma(z_{nk}) = \frac{\pi_k N(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j N(x_n|\mu_j, \Sigma_j)} \tag{10}$$

2. M Step: Update π_k, μ_k, Σ_k:

$$\begin{aligned} \mu_k &\leftarrow \frac{1}{n_k} \sum_{n=1}^{N} \gamma(z_{nk}) x_n \\ \Sigma_k &\leftarrow \frac{1}{n_k} \sum_{n=1}^{N} \gamma(z_{nk})(x_n - \mu_k)(x_n - \mu_k)^\top \\ \pi_k &\leftarrow \frac{n_k}{N} \end{aligned} \tag{11}$$

5.2 Experiment Details

The model performance of GMM highly depends on the features of data we extracted. I tried two kinds of features:

1. HOG feature extracted by hog algorithm with size 324
2. AlexNet Feature extracted by the AlexNet's deep layer with size 1024

Due to that the size of feature we extracted is still really large, it takes a long time to complete the EM algorithm. Therefore, we firstly apply PCA algorithm to reduce the feature to 50 dimension. Then we apply EM algorithm to cluster all the data. Finally, we still use PCA to reduce the feature dimension to 2 for visualization.

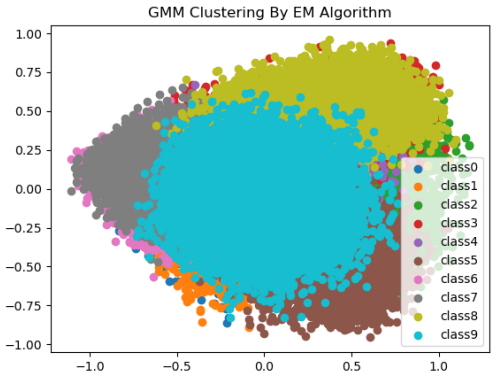
5.3 Empirical Results and Analysis

1. Phenomenon: The visualization of GMM-based classification on hog future is shown in figure 14(a) while the true classification is shown in figure 14(b). From the figure we can see that the GMM performance is not really good because the shape of each class differs the true shape a lot.

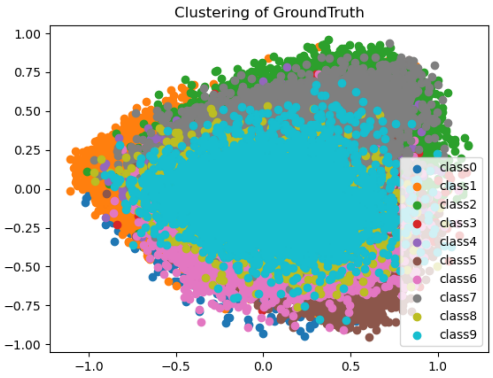
   Analysis: The bad performance is caused by the bad features we extracted according to hog. It is really hard to distinguish each class even in the true classification shown in figure 14(b). How does the GMM succeed?

2. Compared with result of HOG feature, GMM model based on AlexNet feature has really good performance, as is shown in figures 15(a) and 15(b). Generally speaking, the shape and distribution of classes computed by GMM are really similar to the original ones. This shows that the feature extracted by Neural Network is more reasonable and useful.

We can draw some conclusions: the model performance of GMM model is highly dependent of the features input. It can works really well on features extracted by Neural Networks. But I still have a question, if we succeed extracting features based on neural networks, why not just establish the $FullyConnect$ layer to complete the classification task?

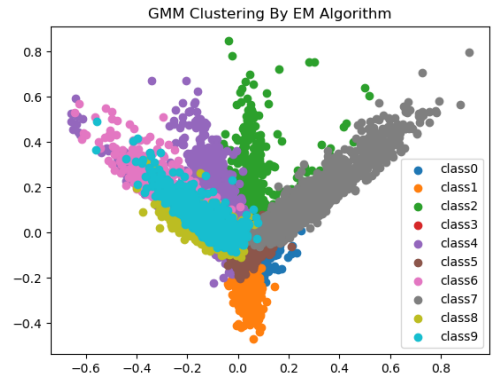


(a) HOG feature: Classification of GMM

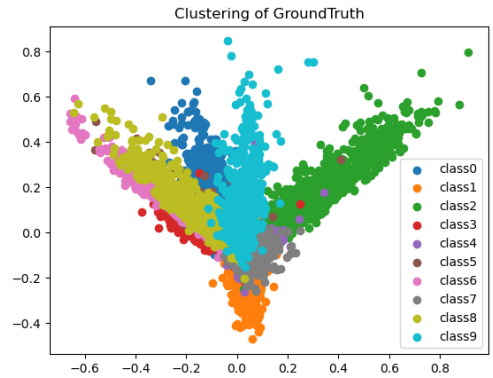


(b) HOG feature: Classification of Ground Truth

Abbildung 14: Hog feature: comparison of GMM and ground truth

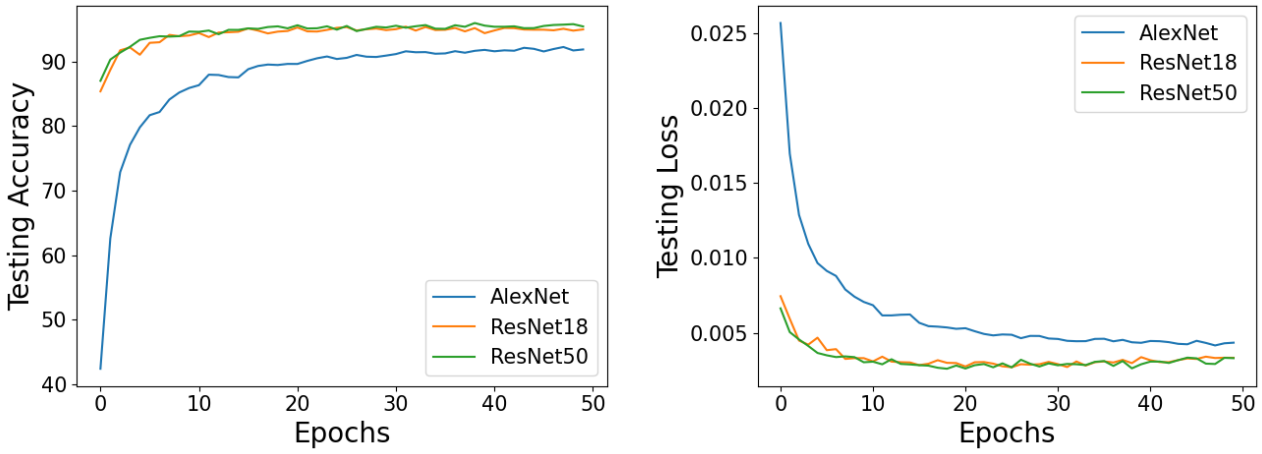


(a) AlexNet feature: Classification of GMM



(b) AlexNet feature: Classification of Ground Truth

Abbildung 15: AlexNet feature: comparison of GMM and ground truth



(a) Testing Accuracy of Neural Networks

(b) Testing Loss of Neural Networks

Abbildung 16: Testing performance of different neural networks

6 Neural Networks

6.1 Introduction

I use three classic neural networks to do this classification task: AlexNet, ResNet18, ResNet50. The input data has size $3 \times 32 \times 32$ which means that the original AlexNet needs to be developed to fit the input size.
The architectures of three models can be seen as follows:

1. AlexNet: $[ConvLayer(in\_channels = 3, out\_channels = 64, kernel\_size = 3, stride = 2, padding = 1)] \rightarrow [ReLU] \rightarrow [Maxpooling(size = 2)] \rightarrow [ConvLayer(in\_channels = 64, out\_channels = 192, kernel\_size = 3, stride = 1, padding = 1)] \rightarrow [ReLU] \rightarrow [Maxpooling(size = 2)] \rightarrow [ConvLayer(in\_channels = 192, out\_channels = 384, kernel\_size = 3, stride = 1, padding = 1)] \rightarrow [ReLU] \rightarrow [ConvLayer(in\_channels = 384, out\_channels = 256, kernel\_size = 3, stride = 1, padding = 1)] \rightarrow [ReLU] \rightarrow [ConvLayer(in\_channels = 256, out\_channels = 256, kernel\_size = 3, stride = 1, padding = 1)] \rightarrow [ReLU] \rightarrow [Maxpooling(size = 2)] \rightarrow [FCLayer(1024, 4096)] \rightarrow [ReLU] \rightarrow [FCLayer(4096, 4096)] \rightarrow [ReLU] \rightarrow [FCLayer(4096, 10)]$
2. I do not change the architectures of ResNEt18 and ResNet50 and thus I do not list their architectures here.

6.2 Experimental Results Analysis

1. The testing loss and accuracy of three networks can be seen in figure 16. Comparing them with the performances of Logistic Regression, SVM and LDA, we can get the conclusion that neural networks do have stronger ability in picture classification task.
2. Hypothesis: In our class, teacher Zhang illustrates that compared with normal Neural Networks, Residual Net can use fewer neutrons to achieve the performance of much deeper network with mechanism similar to Taylor expansion. It not only converges more quickly, but also has stronger ability to learn the residual instead of the direct feature.

   Verification: Figure 16 shows that ResNet converges more quickly in just 8 epochs and finally achiever better performance(lower testing loss and higher testing accuracy) which coincides our hypothesis.

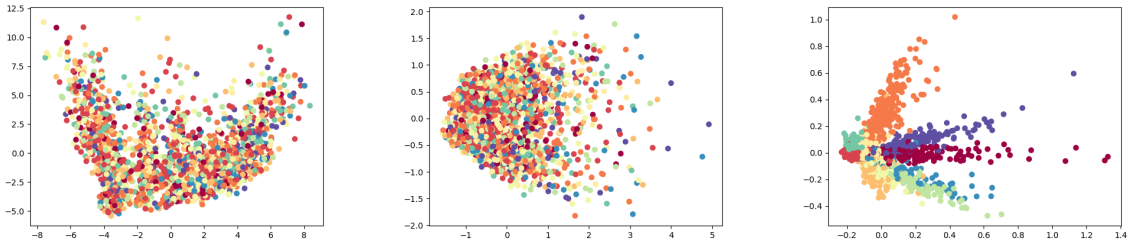
Figure 16 demonstrates that ResNet18 and ResNet50 have nearly the same good performance and both of them works better on SVHN dataset than ALexNet.

6.3 Feature Visualization

In this section, we use three method to visualize the features extracted by AlexNet.

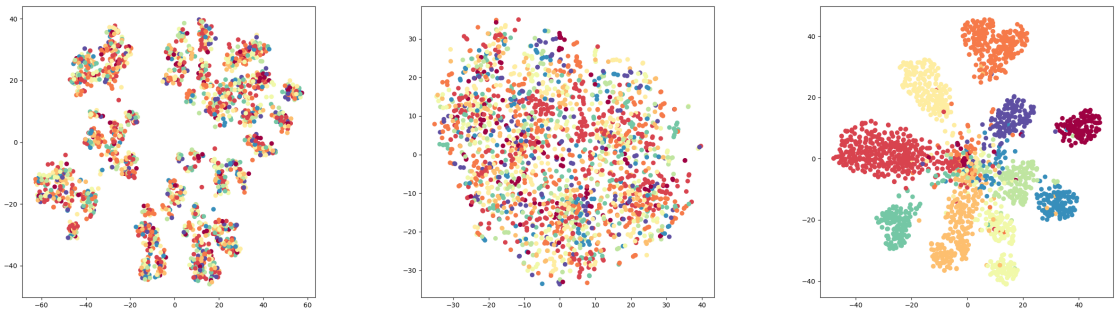
6.3.1 PCA

We randomly choose 5000 data samples and extract their features computed after different $maxpooling$ layers. The flattened output sizes of three $maxpooling$ are 4096, 3072 and 1024. We apply PCA to reduce them to 2-dimensional and visualize them.



(a) PCA: First Maxpooling (b) PCA: Second Maxpooling (c) PCA: Third Maxpooling

Abbildung 17: The PCA visualization of features in different maxpooling layers.



(a) tSNE: First Maxpooling (b) tSNE: Second Maxpooling (c) tSNE: Third Maxpooling

Abbildung 18: The tSNE visualization of features in different maxpooling layers.

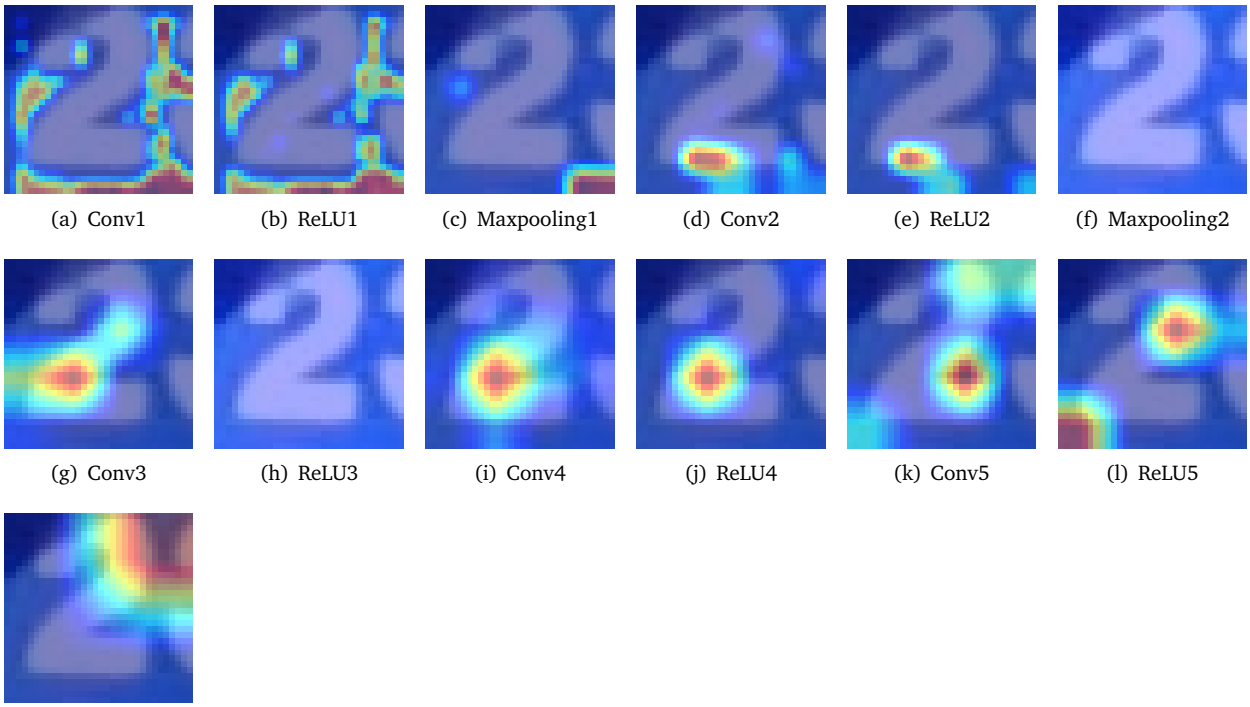
As is shown in figure 17, we can directly see the clustering changes of features from three layers. At the beginning, the data features are dispersed and then the data features of same class will cluster which could be directly seen throug the figure.

6.3.2 tSNE

Similar to PCA, we randomly choose 5000 data samples and extract their features computed after different $maxpooling$ layers. The flattened output sizes of three $maxpooling$ are 4096, 3072 and 1024. We firstly apply PCA to reduce the dimension and secondly use t-SNE to continue reducing them to 2-dimensional and visualize them.
As is shown in figure 18, we can directly see the clustering changes of features from three layers. At the beginning, the data features are dispersed and then the data features of same class will cluster which could be directly seen through the figure. The visualization of tSNE is better than PCA

6.3.3 GradCam

We also use $GradCam$ to show which area in the original picture decides the final prediction class of the input picture. As is shown in figure 6.3.3, we can see that the shallow layers pays more attention to the edges and sides which are distributed and dispersed and deep layers concentrate more on a centralized area.



(a) Conv1 (b) ReLU1 (c) Maxpooling1 (d) Conv2 (e) ReLU2 (f) Maxpooling2

(g) Conv3 (h) ReLU3 (i) Conv4 (j) ReLU4 (k) Conv5 (l) ReLU5

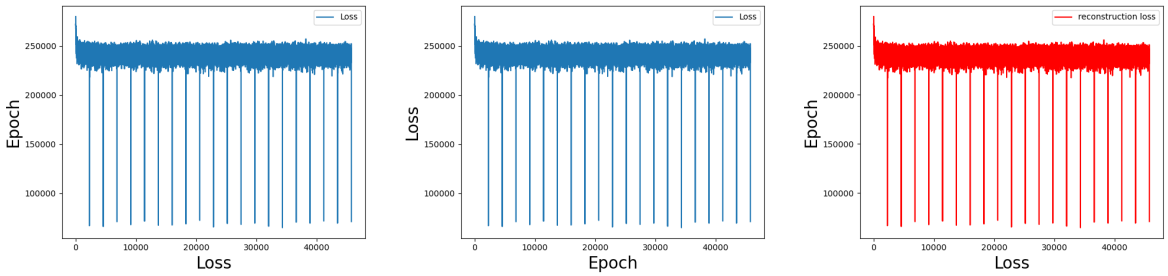
(m) Maxpooling5

Abbildung 19: GradCam: Visualization of areas in the original pictures that activate the neutrons with large weight.

7 Generative Models

7.1 DCVAE

I use the simple DCVAE to complete such generative task.
The training loss can be seen in figure 20.
The reconstruction pictures can be seen in figure 7.1.
I am really sorry that due to the limited time, I have not completed my code to generate good pictures(may exists some mistakes). I will complete this task in the summer vacation.



(a) Loss (b) KL Loss (c) Reconstrution Loss

Abbildung 20: VAE: loss

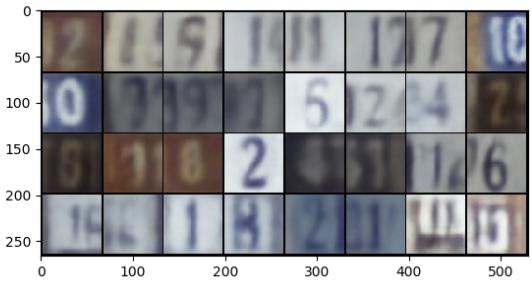


Abbildung 21: VAE: Reconstruction Pictures

7.2 GAN

I use the original DCGAN to complete the generation task.
The loss of discriminator and generator can be seen in figure 22.
The generated pictures can be seen in figure 23. The effect is not really good but you can still see that they are actually numbers. I think that I can improve it in the summer when I have enough time.
Thank you for reading my report! Many Thanks!!!

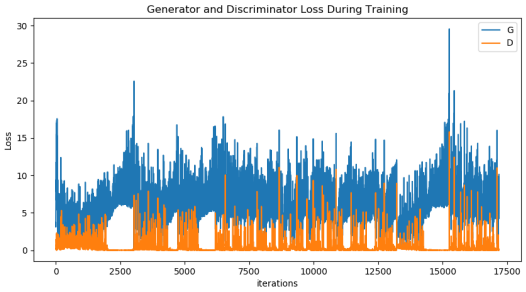


Abbildung 22: Gan: Loss

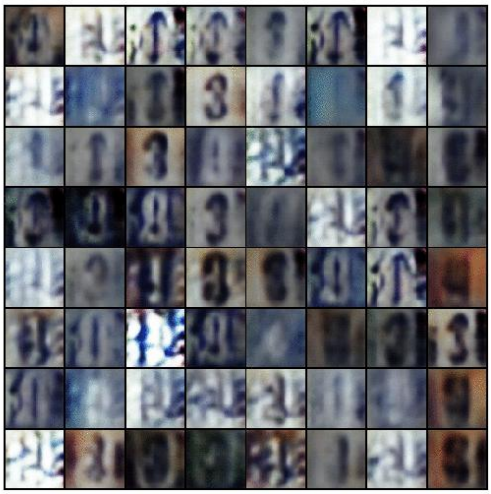


Abbildung 23: GAN: Generated Picture