Projection and Transformation: How to Construct a Good Model

DiXi Yao

Computer Science, {Jimmyyao18}@sjtu.edu.cn

***Index Terms*—Machine learning, Linear models, RNN,CNN, VAE**

I. Introduction

In this project, I implement different machine learning models including basic linear models, non-linear models and neural networks. During each part, I have some extensions beyond the basic conclusion we can derive. After implementing these models, I also use some visualization methods to diagnosis the models and examine the parameters learned by these models. Some small experiments are conducted to verify my guesses and conclusions. Besides having a deeper understanding of machine learning, I generate some my own understanding and conclusions. I would like to share three conclusions which I think is the most interesting after I completing this project.

1) When next time talking with others, we had better not use the flops or parameters to desrcibe the complexity of a model. Instead, we can use how many non-linear functions the model need to desribe it.
2) It is very suprising that to some extent RNN is just the same as CNN theoretically.
3) We can conclude a model as the combination of projection operations and transformation operations. A good model is a perfect arrangement of various projections and transformations.

In this project, we use the SVHN dataset. The SOTA performance of this dataset is 99.0% accuracy by WideResNet-28-10, 98.41% by DenseNet, 66.55% by TSVM and 77.93% by KNN according to paperwithcode [1]. I will use these as some references of my performance. In the rest of the report, the result will be shown in accuracy but % is omitted.

II. Basic Linear Models

In this section, I implement several different linear regression models to solve the classification problem and generate some my own understanding. The result of basic logistic regression model is shown in the table I. Without particular explanations, in this section all models are trained for 10 epochs over all training data. The optimizer is SGD and the learning rate is 0.001 and the momentum is 0.9.

[1]https://paperswithcode.com/sota/image-classification-on-svhn

A. Logistic Regression

The logistic regression is the most basic linear regression model, it uses the sigmoid function to transform the regression results into the classification result. I use torch.nn.Parameter to represent W and the loss function is

$$L = \log(1 + e^{-Y^* X^T W})$$

. I implement two different logistic classifiers, the first one is composed of 10 classfiers and the second one is a one class multiple-class classifier. I extend the binary logistic regression to the multi-class lgistic classification. The one class multiple-class classifier is implemented with W as 3072×10 size and the losses is calculated as

$$L = \sum_{k=1}^{K} \log(1 + e^{-Y_k^* X^T W_k})$$

. Since the multiple classifier can learn the relate information and try to discrete between other 9 classes from losses. The performance is better.

B. Ridge Regression

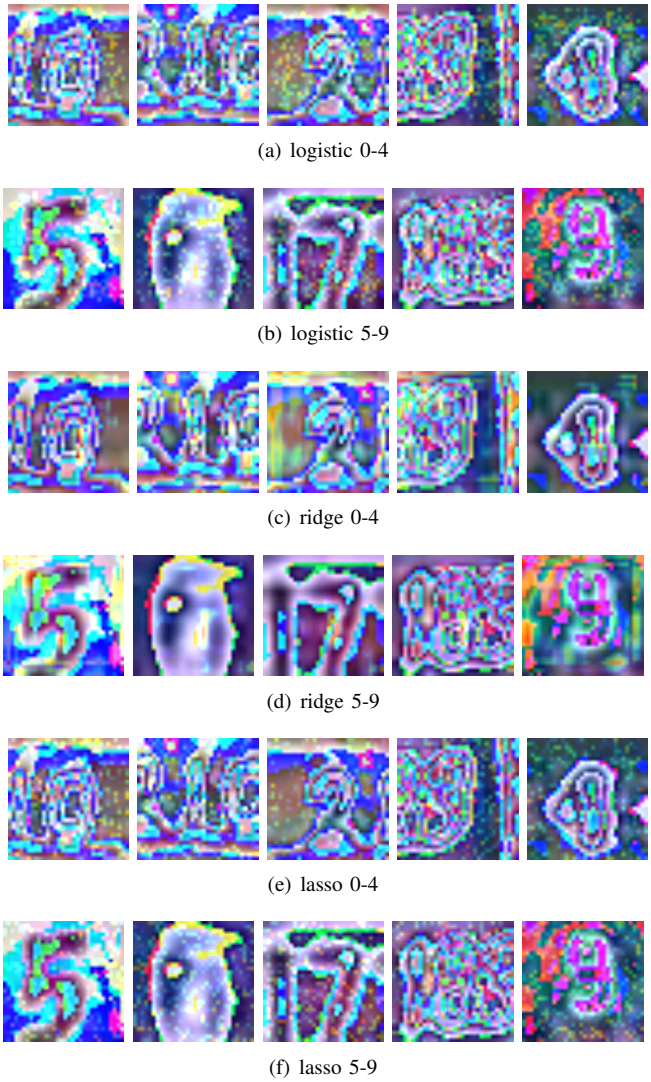
Then I change the loss functin of logistic regression into

$$L = ||Y - X\beta||^2 + \lambda||\beta||^2$$

. The ridge regression use a parameter λ to give restrictions on β, and prone to equally use all features. As shown in figure 1, I trained 10 classifiers and for each classifier and take out the weights. Since the weights is 3073 dimension, I visualize the each dimension parameter on the corresponding pixels in images. For example, the color at image(1,1) is correspoding to β_1. In the image, we can see the models have some ability to capture the attention and learn some weights. Comparing to logistic regression, the heat map is more smooth, because all features are equally used. From the

TABLE I
The result of basic logistic regression models on SVHN

	10 binary	1 multiple	
logistic	16.48	18.42	
λ	1	0.1	10
lasso	11.11	10.83	12.93
ridge	13.35	11.48	13.88
λ		10	
spline		19.60	



(a) logistic 0-4

(b) logistic 5-9

(c) ridge 0-4

(d) ridge 5-9

(e) lasso 0-4

(f) lasso 5-9

Fig. 1. The visualization of W weights on the original images.

result, different λs have diffren tresult, larger λ lead to smaller weights, smaller weights lead higher sensitivity to features which expand noises. Especially, in SVHN, the background information are all noise.

C. Lasso Regression

Comparing to ridge regression, lasso regression changes no dominating features into sparse features through thresholding. The loss function of lasso regression is

$$||Y - X\beta||_{l_2}^2 + \lambda||\beta||_{l_1}$$

. Larger λ can lead to sparser features. In figure 1, we can see that the weights learned by lasso regression is sparse and the white points in figures mean that the correspoding parameters are 0. As a result, larger λ can give more attention to critical information which yields higher performance.

D. Spline Regression

Previous models all standing on the point of a linear regression with β. While such linear transformation may fail to capture the real boundary behind data. Spline regresson introduces the ReLU function to do linear regression The loss functoin is

$$\sum_{i=1}^{n} ||y_i - \alpha_0 - \sum_{j=1}^{P} \alpha_j max(0, x_i - k_j)||^2 + \lambda \sum_{j=1}^{P} \alpha_j^2$$

. We can write the result $\hat{y}$ in another form

$$\hat{y_i} = \sum_{j=1}^{P} \alpha_j ReLU(x_i - k_j)$$

. As a result, if we use 3072 knots, then it will be composed of 3072 ReLU functions. With the introduction of non-linear function, the spline regression can achieve good performance which certrains the necessaity of non-linear transformation in processing high-dimension data.

E. Overfitting and Underfitting

In the experiemnt, I find that all models suffer from either overfitting or underfitting, or both. For basic logistic regression, the overfitting problem is critical since there is no porper control over the magnitudes of weights. While for lasso regression and ridge regression, no obvious overfitting is found since regularization is a good method. For spline regression, the overfitting exists because the knots nums can cause great effect on that.

F. Summary

Professor Zhang introduced the feature capability of different linear regressions in lectures. The ridge regression has no dominating features and the lasso regression has sparce features. To verify such conclusions, I implement these models and design an experiment with setting different λ. Also I visualizae the learned parameters. The experiment shows that the conclusion is basically correct, but also point out that for images with noise sometimes sparce features are better while ridge regression may lose effects.

Besides that, I also investigate the problem of overfitting and underfitting. Though experiments, I have an understanding of how these models deal with overfitting and promise the regularization methods.

III. Support Vector Machines

TABLE II
THE RESULT OF SVM ON SVHN

	linear	poly,d=2	poly, d=3	sigmoid	RBF
Direct	16.11	36.60	-	15.31	30.26
Feature Extracted	88.61	19.59	19.59	89.19	89.26

Support vector machines represent another idea of solving the classification problem. Instead of learning from losses, SVM learns from margin. The objective of the classifier is

$$\min_{w,b} \frac{1}{2}||w||^2$$
$$s.t. \quad \forall i, -y_i(w^T X_i + b) + 1 \leq 0$$

In the actual implementation, I train ten binary classifiers to do the multiple classfier problems. Due to the limited computation ability, I only use 10000 pictrues for traning the SVM.

A. Direct Support vectors

I first direcly flatten the image into 3072 dimension vectors and use the vector to do the classfication. Here, I use four different SVM model: linear kernel, polynomial kernel, sigmoid kernel and RBF kernel. The result is shown in table II. The linear kernel and the sigmoid kernel is relatively low and the polynomial kernel and RBF kernel achieves higher accuracy.

The interesting thing is the linear kernel has the same performance as RBF kernel. However, as we have discussed in last section, this problem is a non-linear classification problem, but the linear kernel still can solve very good or even better than the RBF kernel. Actually, let us revise the kernel based SVM, we use $K(X_i, X_j) = \phi(X_i)^T \phi(X_j)$ to project the features into a higher space in which make the features can be linearly classfied. And 3072 is already a high dimension, so the projection is useless, or even worse. That is also the reason why polynomial and sigmoid fails, because they project the noise but omit the important features.

Then the interesting problem is why RBF does not fail. Actually linear kernel is a special case of RBF kernel. First, we rewrite the linear SVM into kernel form, where

$$\phi(X_i) = \sqrt{C} X_i$$

$$K(X_i, X_j) = C X_i^T X_j$$

. Then we use taylor expansion for RBF kernel

$$K(X_i, X_j) = 1 + \frac{X_j^T X_i}{\sigma^2} + \frac{(X_j^T X_i)^2}{\sigma^4} + \frac{(X_j^T X_i)^3}{\sigma^6} + \cdots$$

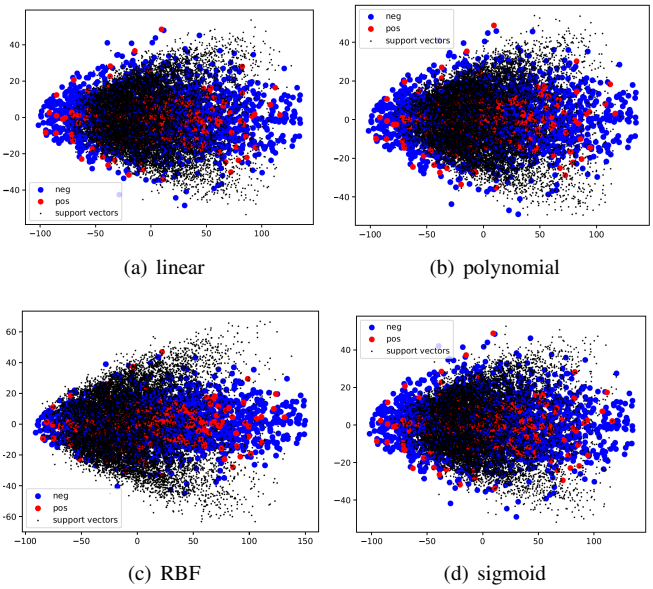
As a result, if we set a relatively large σ. Then C in linear SVM will be equal to $1/\sigma$ and the rest is small so we can omit. In this case the linear SVM is actaully a sepcial case of RBF kernel SVM. Actually it has been already proved in [1], the proof is more strict and general. To validate it, I set σ as 1000 and the classification result is **34.41%** which is close to linear kernel. More deliberate setting can improve the performance further

If we carefully examine the explanation of RBF kernel, we can see with a carefully selected σ it can also represent polynomial kernel. In [2], it is also proved that RBF can represent sigmoid kernel. The only thing matters is how to choose σ.

To better understand the representation of models, I also use PCA to flatten all samples on a 2D plane. I choose the 9th classifier for demonstration in the report. It is shown in figure 2.The support vectors of RBF is prone to be more separate and the support vecotrs of linear is more close to the boudary of different samples. The polynomial and the sigmoid is a little bit of overfitting.

B. Support vectors for extracted vectors

Since, we have discussed that SVM can project features to linear space or directly process features in linear space. I used the 1024 dimension features extracted by neural networks for better understanding of SVM's classification ability. Since the final layer of nerual network can process these features, it should be a linear divisble problem. The result shows that three types of kernel has a good classification. But the ineresting and confusing thing is the polynomial performs bad and even worse than on unprocessed data.



(a) linear (b) polynomial

(c) RBF (d) sigmoid

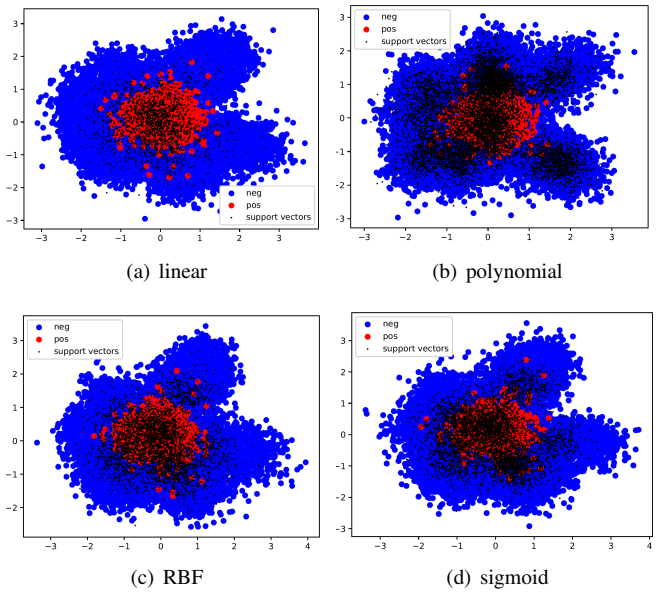
Fig. 2. The visualization of support vectors on 9th class

One reason is due to the **numerical instability**. When $(X^T + c) < 1$, it will closer to 0 with increasing d and when $(X^T + c) > 1$, it will closer to ∞ with increasing d. As a result, the trainig process may become relatively unstable. Another problem is that as we have discussed, RBF and sigmoid can have special cases into linear kernel while polinomial kernel does not have such property. With the linear divisble problem, we use a curve to fit it which can both lead to underfit and overfit. This is also systematically explained in [3]

The visulization of vectors is shown in the figure 3. We can see three types of kernel can dicerete well. But polinomial kernel fails to generate good support vectors. We may falsely interpret numerical stability problem as overfitting. But in my experiment, the training acc is 18.49% which means no overfitting occurs. In the subfigure b), we can see the support vectors prone to separate at every places, not at the real boundary. This is because polynomial fails to depict the linearity. If our dataset is with two-dimension features, we cannot use $\frac{x^d}{a^2} + \frac{y^d}{b^2} = c^2$ to fit $ax + by = c$ with $d > 1$ and $a, b, c > 0$.

C. SVM summary

By implementing SVM, we learn the classification based on confidence, it is similar and related to linear regression if we view the margin loss as hinge loss. However, the linear regression project a high dimension space into a low dimension space which is linear divisible. While, SVM project a high dimension space into a higher dimension space which is linear



(a) linear (b) polynomial

(c) RBF (d) sigmoid

Fig. 3. The visualization of support vectors on 9th class with extracted features

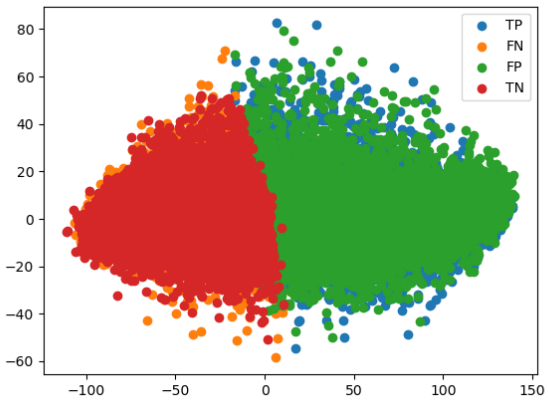


Fig. 4. The visualization of TP,TN,FP,FN of LDA classification results. TP+FP means the predictive positive samples. TP+FN means the actual positive samples.

divisible. The RBF is flexibility as it can write into other forms of kernel, but the choice of σ should be careful. Linear kernel will be easier implemented and set which is still a good choice currentlly

IV. Linear Analysis Method

A. LDA

LDA is a linear classfier which can maximize the inter-class variance and minimize the intra-class variance. I first directly implemented the LDA to calculate $\mu^+, \mu^-, \Sigma^+, \Sigma^-$. Then I calculate the matic $S_w^T S_b$ and extract the eigenvectors. I choose the vector with the largest λ and it shall be 3072 dimensions. Then I multiply it with each sample and get a number. The number is bigger than 0, it is positive otherwise it is classified to the negetive. However, the result is only 11.42% which means almost no knowledge is obtained. Then I use it to solve a 2-class classification. $0 - -4$ is negetive and the rest is the positive and the result is 49.42% the same as random guessing. Then I guess the LDA fails to capture the eigen vector and I check the biggest λ, and actaully all λ_e almost take 100% portion of all the sum of λ. The result of LDA is shown as figure 4. We can see the boudary of TP+FP and FN+TN is obvious. So, actually LDA gives a obvious boundary but the samples are too high dimensioned to be divided. As a result, LDA cannot solve such problem

B. Mixture of Gaussians

Mixture of Gaussians is an estimator using gaussian distributions to fit any possible distribution. It is a supervised classifier. I also use the calculation method encapsulated in the pytorch to do such work. The gaussian model can be write in

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

. As a result, there are three parameters μ, Σ, π. I use the EM algorithm to optimize such parameters. In the E step, we calculate estimate ptobability with such equation

$$\log(\gamma(z)) = \log \pi_k + \log \mathcal{N}(x|\mu, \Sigma) - \sum_{j=1}^{K} (\log \pi_j + \log \mathcal{N}(x|\mu, \Sigma))$$

. In the M step, we update the parameters based on the γ in E step.

I use the 1024 dimension vectors generated by neural network to do the GMM. However, the classification result is 47.22%. Much lower than only one layer of linear classification. With a given feature x_i and the probability of class k, the representation of linear layer is

$$sigmoid(x_i^T w_k + b_k)$$

and the representation of GMM is

$$\pi_k e^{\frac{||x_i - \mu_k||^2}{2||\Sigma_k||^2}}$$

I visualize the inter features I used by PCA+TSNE in figure 5 and we can see actaully the distribution of each class is not strictly the gaussian models. Theoritically more gaussian kernels are needed to fit such distribution and only $k = 10$ is not enough. This is the main reason of the failure of GMM.

V. Neural Networks

Since simple linear or non-linear transformation and regression cannot well solve the problem. I further my step into the nerual networks.

A. Multi-Layer Perceptron

I implement the MLP through combination of linear layers and non-linear functoins. The structure of MLP is shown as table III. The learning rate is 0.01 and is set to 0.001 after 30 epochs. The model is trained for 40 epochs and the batch size is 256. I use the SGD optimizer with 0.9 momentum and 1e-09 weight decay. The classification accuracy of MLP is

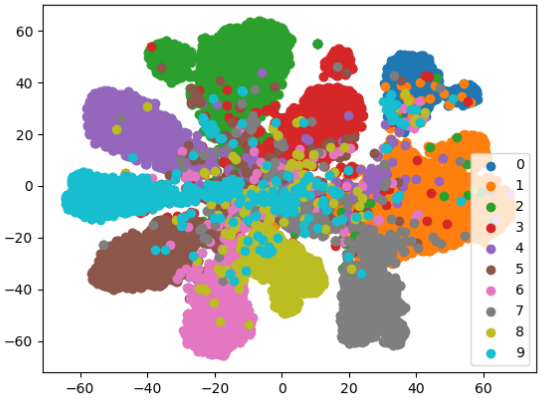


Fig. 5. The TSNE visualization of features gemerated by neural networks. First use PCA to 10 then use TSNE to 2.

TABLE III
THE MODEL STRUCTURE OF MLP

Linear(3072,1000)
ReLU
Dropout(0.2)
Linear(1000,1000)
ReLU
Dropout(0.3)
Linear(1000,1000)
ReLU
Dropout(0.4)
Linear(1000,1000)
ReLU
Dropout(0.5)
Linear(1000,10)
Sigmoid

78.25. Actually, we can view the MLP a combination of ridge regression and spline regression with multiple layers. MLP is actaully an ultimate combinatin of good linear methods. However, it still cannot compare to good performance.

Then I use guided backpropagation to do the visualization of how the model classify the model. The visualization is shown as figure 6. We can see for most of the number, MLP can alreadt have the good discriminations. However, there still exists some cases that the MLP cannot recognize correctly.

(a) original image (b) guided backpropagation

Fig. 6. The guided backpropagation visualization of MLP

From the experiments, we can conclude that the MLP has already been capable of solving high dimension problems theoretically. However, due the fully connected property of the MLP, there are too many parameters need to optimize. The model is too complex to optimize and it is too difficult to find the solution point of the MLP. As a result, new techiniques are needed.

VI. CONVLUTIONARY NEURAL NETWORKS

TABLE IV
THE CLASSIFICATION RESULT OF CNNS

	MobileNetV3	NiN
ACC	95.89	95.60

CNN is definitely a great invention from 20th to 21st century. I aslo implement two CNN models in this project. ResNet, AlexNet and VGG these classic model structure will be detailed and fully examined in project2. In this project, I try two a little bit different but interesting model: MobileNetV3 [4] and NiN [5]. The training result is shown as table IV.

A. Model Implementations

1) MobileNetV3: The structure of model can be reffered to MobileNetV3 small in [4] except that I do some modifications to let the network can be fed in 32×32 pixels images and give out 10-class classification. The network is trained for 40 epochs with learning rate 0.1. The learning rate is set to 0.01 at epoch 30 and the optimizer is SGD with momentum 0.9 and weight decay 1e-9. Except different layer numbers, layer channels and skip connections. There are two interesting things, the fisrt is the 1×1 convolutional kernel which we will discuss it later, and the second is the squeeze and excite structure [6]. The structure of SE-inception unit is shown as figure 9. It can written in the form

$$h = x * s = x * \sigma(W_2 ReLU(X^T W))$$

Such activation gives different weights on the features, which let the network can have different attention and activation on different parameters. The idea is insteresting because it let the network learn from itself.

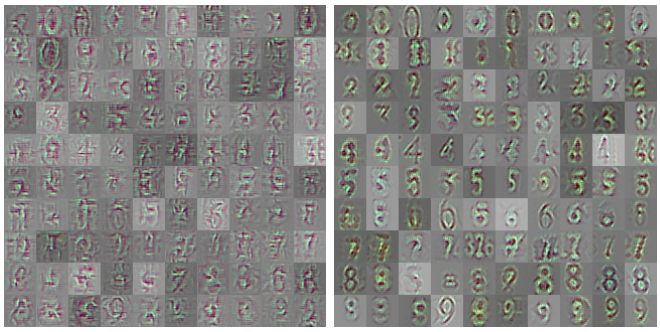
TABLE V
THE MODEL STRUCTURE OF NIN

Layer(in,out,kernel,padding,stride)
ConvReLUBN(3,192,5,2,1)
ConvReLUBN(192,160,1,0,1)
ConvReLUBN(160,96,1,0,1)
MaxPool(k=3,s=2,p=1)
Dropout(0.5)
ConvReLUBN(96,192,5,2,1)
ConvReLUBN(192,192,1,0,1)
ConvReLUBN(192,192,1,0,1)
MaxPool(k=3,s=2,p=1)
Dropout(0.5)
ConvReLUBN(192,192,3,1,1)
ConvReLUBN(192,192,1,0,1)
ConvReLUBN(192,10,1,0,1)
AvgPool(k=8,s=1,p=0)

2) Network in Network: The structure of model is shown as table V.The network is trained for 40 epochs. The learning rate deacys 10 at epoch 30 and the optimizer is SGD with momentum 0.9 and weight decay 1e-9. The learning rate of different layers are differet. The learing rate for final layer is 0.1 and 1 for others.

The most core and interesting idea in NiN is that it does not have fully connected layers. If we have a feature $h \in R^d$. Then we can use a convolutinary layer with r 1×1 kernels and get a $h' \in R^r$ features where $h'_j = \sum_{i=1}^{d} w_{ij} h_i$ for $j = 1, \cdots, r$ while for fully connected laters with $W \in R^{d \times r}$. $h'_j = \sum_{i=1}^{d} w_{ij} h_i$ too. So, in the formulation, they are completely equal. But, the 1×1 conv can be inserted in the inter layer of CNN model. As a result, after normal conv, we have this special kind of mlpconv, which means it acts like MLP. In this elegate form, we compose two 'weapons', MLP and CNN in one model and reach the good performace.

B. Why CNN succeed



(a) MobileNetV3 (b) NiN

Fig. 7. The guided backpropagation visualization of MobileNetv3 and NiN. The original graph can be referred to figure 6

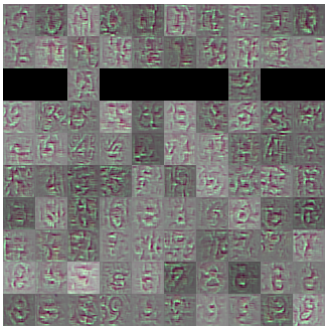


Fig. 8. The guided backpropagation visualization of MobileNetv3 without SE unit. The original graph can be referred to figure 6

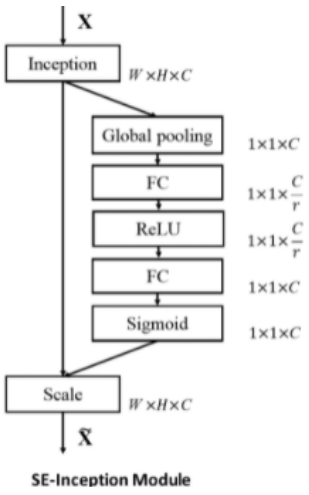
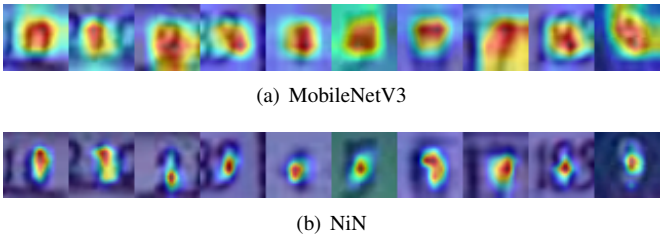


Fig. 9. The structure of SE unit

1) Attention Analysis: First, I use the guided backpropagation to visualize how two models capture the critical information in the images. As shown in figure 7. The $\frac{\partial L}{\partial X}$



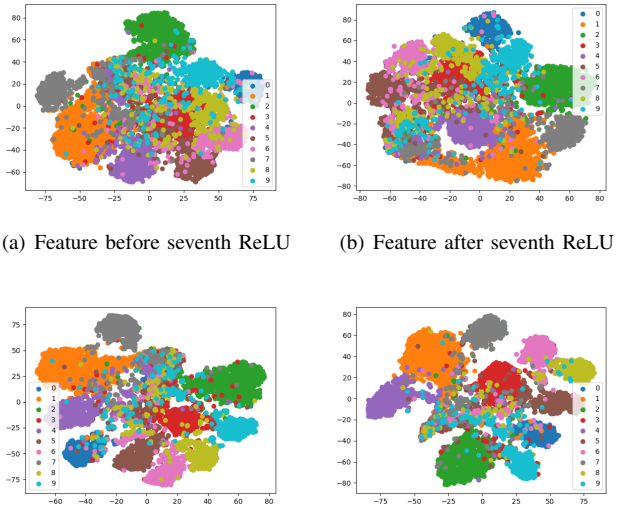
(a) MobileNetV3

(b) NiN

Fig. 10. The gradcam of features before final avrage pooling.

generated by NiN is closer to the number in the picture. So I try to investigate why NiN can generate the better activated boudaries. First, I think it may caused by the SEunit, so I deactivate them and the result is shown as figure 8. So, it might not possibly caused by that. Through the visualization of GradCAM (figure 10), the attention of the NiN is more centrelizaed on the critical part while the mobilenet fail to concentrate. MobileNet still may pay attention to some interelevant information in the background. But comparing to MLP, convolutinary layers can have a good feature compression and extraction ability.

2) Feature Analysis: Then I use the PCA(m=10) and TSNE(n=2) to visualize and analysize the intermediate features generated by CNN. The feature before fully connected layer in MobileNet is shown in figure 5. The feature before final average pooling layer in NiN is shown in figure 13(f). The CNN can give a good representatoin of intermediate features.

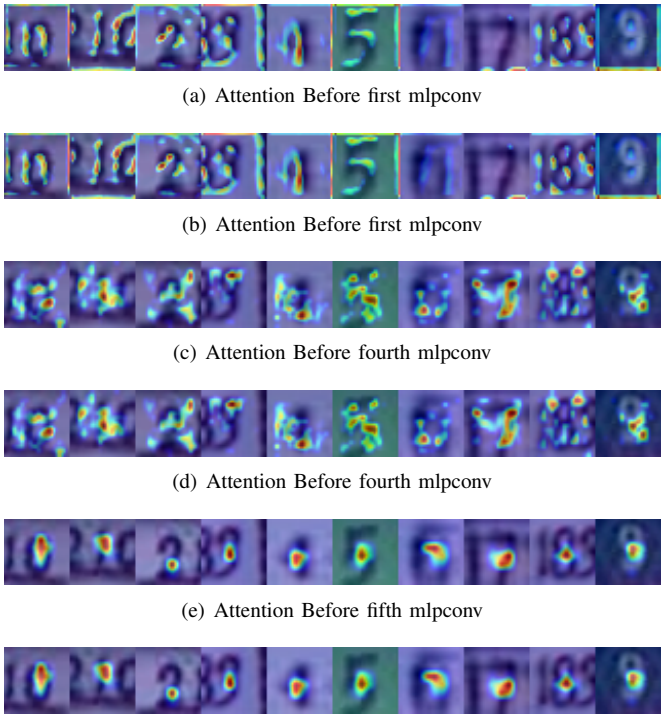


(a) Feature before seventh ReLU (b) Feature after seventh ReLU

(c) Feature after eighth ReLU (before is (d) in figure 13) (d) Feature after nineth ReLU (before is (f) in figure 13)

Fig. 11. The feature before and after ReLU function with PCA(m=10)+TSNE(n=2) visualizatoin

I also want to investigate how non-linear transformation contributes to the discrimination abilities of a CNN model. The visaualization of feature before ReLU function is shown in figure 11 and we can see that suprisingly the relu does not have direct impact on discrimination ability. As a result, the non-linear function contributes to the transformation but not the projection.



(a) Attention Before first mlpconv

(b) Attention Before first mlpconv

(c) Attention Before fourth mlpconv

(d) Attention Before fourth mlpconv

(e) Attention Before fifth mlpconv

(f) Attention Before fifth mlpconv

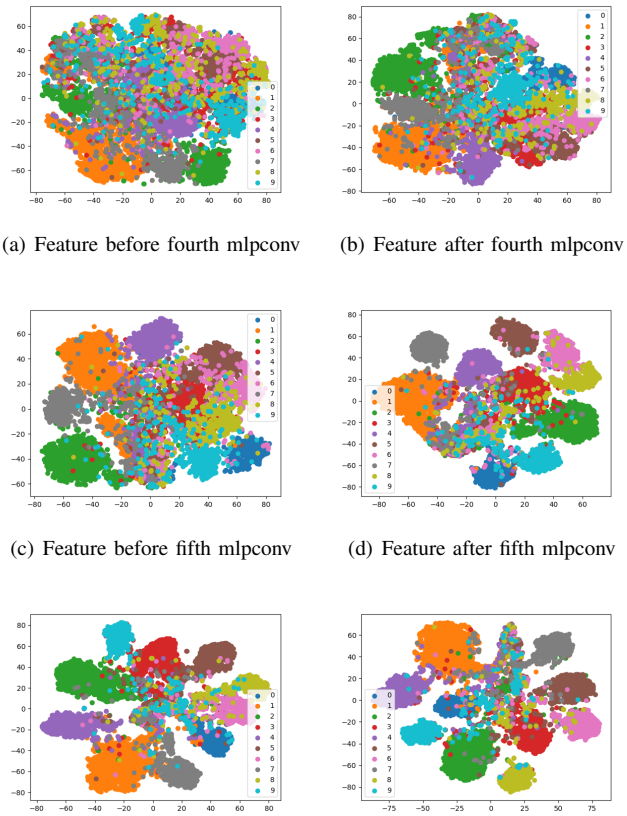
Fig. 12. The attention map before and after MLPConv.

3) MLPConv: MLPConv is an instresting idea and I would like to further investigate why such method can achieve success. So I conduct experiments to check what mlpconv contributes to. From figure 12, the attention map changes almost nothing before and after the MLPConv which means that convolutional layers has good ability of capturing attention and MLP's ability to capture attention is far more inferior to the convolutional layers. When they are combined together, MLP does not contribute to gather attention.

Then I use PCA+TSNE to analysize the intermediate feature before and after doing the MLP conv in figure 13. After each MLPConv, the tSNE and PCA can better disrciminate the high dimension features. As a result, the MLP use a linear transformation to project the original space to a better space which is more fit for non-linear transformation. Such projection is just the perceptron where we feel an entity and convert it into our feelings which is a better space. As a result, we can understand that MLPConv is combination of compression and preceptron which makes it a good projection for further transformation.

C. Complexity

Through implementing different models, I found that the how the model is non-linear can greatly decide the performance of the model. The NiN succeeded in introduce mlp into CNN and gives it more chance of doing non-linear transformation. MobileNet use the SEunit which also introduces much non-linear transformation through squeeze and excitation. Spline regression can also prove such conclusions. Actually, in [7], it has been pointed out that for the feature of a specific intermediate layer, we define the real complexity of this feature



(a) Feature before fourth mlpconv

(b) Feature after fourth mlpconv

(c) Feature before fifth mlpconv

(d) Feature after fifth mlpconv

(e) Feature before sixth mlpconv

(f) Feature after sixth mlpconv

Fig. 13. The feature before and after MLPConv with PCA(m=10)+TSNE(n=2) visualizatoin

TABLE VI
THE RESULT OF USING DIFFERENT NUMBER OF NON-LINEAR FUNCTIONS IN NIN

# non-linear functions	ACC
3	86.93
4	85.67
5	93.24
6	94.50
8	95.46
9	95.60

as the minimum number of nonlinear transformations required to compute this feature. In NiN, I investigate the last feature of the network before average pooling and try different number of nonlinear functions. The final accuracy with different functions is shown as tabel VI. With increasing number of non-linear functions, the model can yield higher accuracy which verfies how the model's non-linearity determines how good the model is.

1) Conclusion: There are two basic operations in a model, the projection and the transformation. The projection is to project the features in one space into another space but does not change the interrelationship between samples. For a projection component, we can model it as

$$q_\phi(X^T W|X)$$

. The transformation is to transform the original features into new features which may break the previous relationships. For a transformation component, we can model it as

$$f(X)$$

where f is a non-linear function. A good model is a combination of reseaonable arrangements of different projection and transformation.

CNN is an example, the projection in CNN includes: convolutional layers, linear layers, bacthnormalization layers and the transformation includes: pooling layers and non-linear activation functions. The proper combinations of them create a good model. If we have a feature $C \times H \times W$ and we use R conv kernels with K kernel, then we have RK^2 parameters. We assume the output feature is still $R \times H \times W$, it will cost linear layer RCH^2W^2 parameters. As a result, standing on three points

1) Standing on the point of compression. The conv layer can have a good compression ability while linear model's compression is theoretically the same as conv but actually far worse, because too many parameters cause linear model easier to focus on noise.
2) Standing on the point of extraction. Both layer are linear layers so they have the same linear transformation ability. However, in the CNN, we can insert far more non-linear transformations which are very critical.
3) Standing on the point of optimizatoin. Conv layers has far more less parameters than linear layers which lead the CNN is far more easier to optimize especially when the model becomes deeper.

In [8], it is pointed out that deep learning is the simulation of human learning. As human we have two important processes: taking courses (also reading books, watching videos) and thinking ourselves. Taking courses is like projection, to project others' good knowledge in our mind. Then thinking ourselves is like transformation, to make the knowledge as our own knowledge. There also exist other perfect explanations about how CNN workds [9], [10] and the key knowledge behind them is to explain why the machine stuff is like our human minds.

VII. Recurrent Neural Network

RNN is another classic type of nerual network. Though it is used for processing temporal information commonly, I also try to use the RNN to solve SVHN task and analysize what is the difference between RNN and CNN models. Here, I give a brief introduction and how I implement them. I implement two RNN models, one is vanila RNN model where each cell is

$$s_t = f(Ux_t + Ws_{t-1})$$

$$o_t = g(Vs_t)$$

Another model is LSTM which is designed to overcome the vanishing gradient problem in vanila RNN models. The main difference is memory cells in LSTM.

To make it feasible to train SVHN, I flatten the image into the size of 32×96. This form means that I take each row of image with RGB (3 channels) as a timestamp. The row1, row2 to row32 correspoding to the $x_1, x_2, \cdots, x_{32}$ in RNN model. For each timestamp example, the feature is 32 dimension. Different from CNN processing through one square block one after another, RNNs process one row after another. The dimension of hidden vectors is 128. Then a fully connected layer is used to project the hidden vectors on 10-class classification plane. With no particular explanation, I train the models in this section with 60 epochs. The learning rate is 0.1 and divded by 10 at 30,40,50 epochs. The optimizaer is SGD with momentum 0.9 and weight decay 1e-9. The running result is shown in table VII

TABLE VII
THE REUSLT OF DIFFERENT RNN MODELS ON SVHN

	ACC
vanila layer1	18.93
vanila layer3	22.07
vanila layer3 bi	21.63
LSTM layer3	90.86
Bi-LSTM layer3	90.71
Bi-LSTM layer5	90.30
LSTM layer3 dropout	90.16
LSTM layer5 dropout	89.29
LSTM layer10 dropout	19.59
LSTM layer3 ReLU	19.59
LSTM layer10 ReLU	19.59

A. Relationship to CNN

First let use see a trivial example, we have a 2×2 image with one channel and we want to use three methods to get the y: linear, convolutional layer and recurrent cell.

1) Linear: $y = \sigma(x_{11}w_1 + x_{12}w_2 + x_{21}w_3 + x_{22}w_4)$
2) Conv: $y = \sigma(x_{11}w_{11} + x_{12}w_{12} + x_{21}w_{21} + x_{22}w_{22})$
3) Recurrent: We assume function f is identity and $V = I$, then it shall be $y = \sigma(x_{11}u_1w_1 + x_{12}u_2w_2 + x_{21}w_1 + x_{22}w_2)$

Then we also can write the RNN into CNN. If we assume function f is identity and $V = I$ and we assume the input is an image $x \in R32 \times 32 \times 3$. Then we can get the output of a recurrent cell as

$$o = \sigma(x_{32}^T \cdot U + x_{31}^T \cdot U * W + \cdots + x_2^T \cdot U * W^{30} + x_1^T \cdot U * W^{31})$$

We also can write it into a summation

$$o = \sigma\left(\sum_{i=1}^{32}\left(\sum_{j=1}^{32}\sum_{k=1}^{3} x_{ijk}u_{jk}\right)w_i^{32-i}\right)$$

$$= \sigma\left(\sum_{i=1}^{32}\sum_{j=1}^{32}\sum_{k=1}^{3} x_{ijk}u_{jk}w_i^{32-i}\right)$$

So, recurrent neural network is also a special structure of fully connected network. And we know we can use multiple convolutional layers to represent a linear layer. As a result, with a particular design, recurrent network is the same as the convolutional netowork. However, the one layer vanila network is only performs as good as a linear regression model (though already verified above deduction) but far from CNNs.

1) More non-linear: One reason for RNN so-so performance may because of it is not non linear enough, so I make the RNN cell into 3 layer deepwise. But the experiment shows that it still fails and the reason is the gradient vanishes after few epochs.

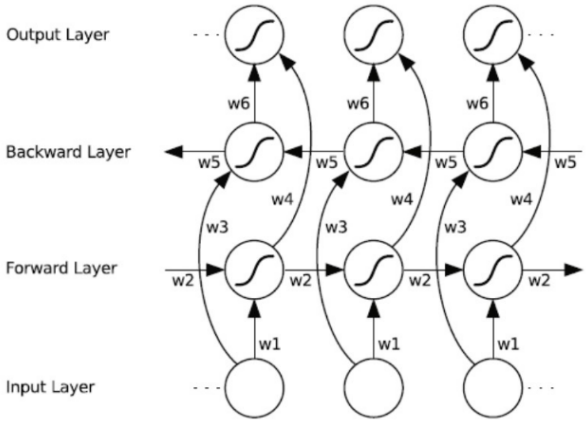


Fig. 14. The structure of Bi-RNN

2) Bi direction: One problem of valina RNN is that W is small number most times and the earlier features like x_1, $x_2 \cdots$ will get a very small weight w_i^{31} which means the information in previous rows will be negelected. Besides, the data in the middle of the image will get smaller weights than those at the bottom of the image which is very irreasonable. A solution is using bi-directional structure [11]. The structure is shown as figure 14. In this way, the features of different layer can share relatively balanced weights. However, the performance is still not good because the backpropagation chain is longer and the problem of vanishing gradient is severer.

3) LSTM: To solve the problem of gradient vanishing, LSTM is introduced. Besides that, the LSTM also has the ability to memorize which means the information in the top rows of image may not be forgotten when recurring to middle rows. And when it coms up to bottom rows, the information in middle rows will not be forgotten. LSTM can achieve good performance.

As a result, I use a Bi-LSTM [12] as the combonitions of these properties. The performance of Bi-LSTM is already commensurate to that of CNN models. Actually, using recurrent network indeed can achieve the same SOTA performance as CNN models. In a github project by lixuan0023[2], he trained a RNN for six days and achieve the test accuracy of **96.34**. So, indeed RNN is the same as CNN to some extent.

The guided backpropagation of RNN is shown as figure 15, we can see there is weak explanation for RNN model because the processing of data is not followed the structure of images. But there are many bands in the images, which demonstrate the logic of RNN processing the data. From figure 16, we can see the model has the good discrimination of features. As a result,

[2]https://github.com/lixuan0023/SVHN_RNN

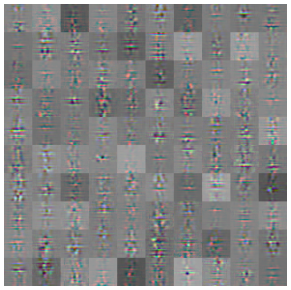


Fig. 15. The guided backpropagation visualization of Bi-LSTM (3 layers). The original graph can be referred to figure 6

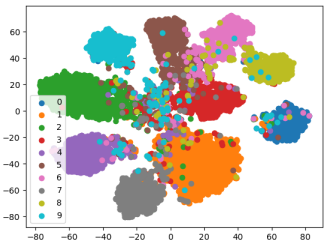


Fig. 16. The PCA+TSNE feature visualization of the 128×2-dimension features generated by the Bi-LSTM

the RNN model is commesurable with CNN model in feature extracting. Standing on this point, they have no difference.

B. Ablation study

Furtherly, I do the ablation experiments to investigate two problems. The first problem is what will happen if the RNN is very deep. [13] gives the answer to why and when deeper is better. I also do some experiments with changing the layer num of LSTM. The result shows that deeper is not better. Reversely, deeper makes it worse since LSTM also occur the problem of gradient vanishing when the model is becoming deeper. The RNN model suffers from the huge parameters and long backpropagation chain when becomes deepr unless other structures are introduced such as transformers [14]. So, not always deeper is better.

The second problem is why the non-linear function in RNN is tanh and sigmoid instead of ReLU family and sigmoid in CNN. Besides, I found that ReLU was firstly invented for solving the gradient vanishing problem in RNN [15], [16]. As a result, I change the activation function f in LSTM into $ReLU$. The result shows that, the LSTM explodes directly and performance is bad. Because if we investigate the formulation of RNN again $s_t = f(Wf(Wf\cdots(W\cdots s_0)))$ which means if a number in W is bigger than 1, then the output will be very large. The relu function will not change the very large output and the result is far out of the boudary. The more sysmetically and detailed explanation can be found in [17]. As a result, we still better use Tanh in RNN. I also change the activation funciton in NiN from ReLU into Tanh, the result is 91.83.

C. Relationship to ResNets

Professor has said that residual network is a special case of LSTM and I would like to investigate this problem. Actually, different from the method of previous implementation, we can implement the LSTM almost the same as the residual networks. For each input x_i of the reccurent cell, we always put in the original graph. For a residual block with n conv layers, the block can be formulated in $o = x + F_n(x)$. Then for a LSTM, we can formulate in the form of $o = x + F_n(x + F_{n-1}(x + \cdots + F_1(x)))$. So the ResNet is a special case of LSTM. However, such implmentation of LSTM can only achieve the performance 51.04. There exist several problems

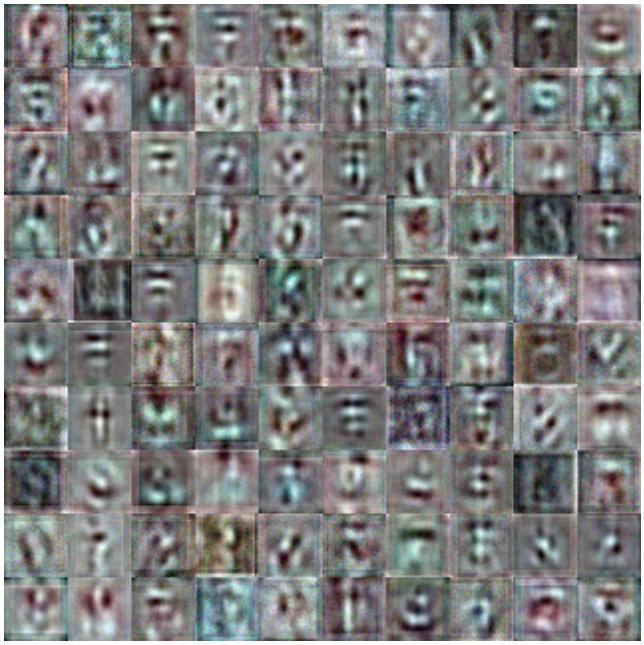


Fig. 17. The guided backpropagation visualization of LSTM implemented following the idea of residual networks

- The problem of projection function. In CNN the projection function is convolutional layer while in RNN it is still a fully connected layer which is very hard for optimization. As a result, CNN can learn better parameters.
- The problem of depth. In CNN, the residual netwrok can be very deep even about 1024 dimension. While LSTM cannot be very deep because of computational resources limitation and gradient vanishing problem.

The visualization of guided backpropagation is shown as figure 17. We can see that the model is sightly capable of depcting the boundary of these numbers. But the boudaries are not clear enough. This can prove that the LSTM can have the theorttical ability as the residual networks but practically optimization is relatively hard.

D. Difference

Though RNN is theoretically the same as CNN, but actually they still have much difference besides their different wide applications.

- Convolution layer is a special and outstanding structure of projection function.
- RNN is hard to become deeper comparing to CNN.
- The different structure gives them different choices of activation function.
- The spatio information can be easily captured by CNN but hard by RNN.

Actually, in many practical problems they are mostly used together instead of using ANN,RNN, or CNN invidialually. The reason behind that different structure has different properties. As a result, a particular architracture which can simultaneously solve all these problems is still waiting for exploration.

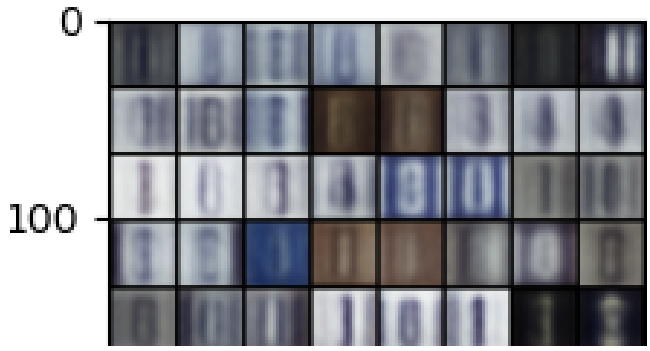


Fig. 18. The reconstructed images generated by vanila VAE

VIII. Variational Autoencoder

Besides discriminative nerual networks, I also learn to use VAE for extract features and reconstruct images. The training framework is similar to training discriminate models. I use 1024 dimension features as latent features. With no particular explanations, the models in this section is trained under Adam optimizer with learning rate 0.001. All training data are used. The loss function of basic VAE is constructed as follows

$$
\begin{aligned}
L(\phi,\theta,X) &= KL(q_\phi(h|X)|p_\theta(h_X)) - E_{q_\phi(h|X)}(\log p_\theta(X|h)) \\
Loss &= E_{q_\phi(h|X)}(\log p_\theta(X|h)) - KL(q_\phi(h|x)|p_\theta(h)) \\
&\quad + KL(q_\phi(h|x)|p_\theta(h|x))
\end{aligned}
$$

We then take a look at these three parts. The first part:

$$
\begin{aligned}
E_{q_\phi(h|X)}(\log p_\theta(X|h)) &= \frac{1}{N}\sum_{i=1}^{N}\log p_\theta(x|h) \\
&= MSE(X,X')
\end{aligned}
$$

The second part:

$$
\begin{aligned}
KL(q_\phi(h|x)|p_\theta(h)) &= \int q(h)\log\frac{q(h)}{p(h)}d_h \\
&= \int q(h)((\log q(h) - \log p(h))d_h \\
&= \int q(h)(\log(\frac{1}{\sqrt{2\pi\sigma^2}}e^{\frac{(h-\mu)^2}{2\sigma^2}}) \\
&\quad - \log(\frac{1}{\sqrt{2\pi}}e^{\frac{h^2}{2}}) \\
&= \int q(h)(\log\frac{1}{\sigma})d_h + \int\frac{h^2}{2}q(h)d_h \\
&\quad - \int\frac{(h-\mu)^2}{2\sigma^2}q(h) \\
&= \log\frac{1}{\sigma} + \frac{\sigma^2+\mu^2}{2} - 1/2
\end{aligned}
$$

The third part is too hard to calculate, so I implement the first two parts in my loss functons and actually, these two parts has the actual representations. The first part represents the ability of the model to shorten the distance of the reality and 'dream'. The second part represents the ability of the model to make dream. It is like we need to have a good dream and try our best to achieve that in real daily life. Follow this loss function, I try to investigate the performance of VAE.

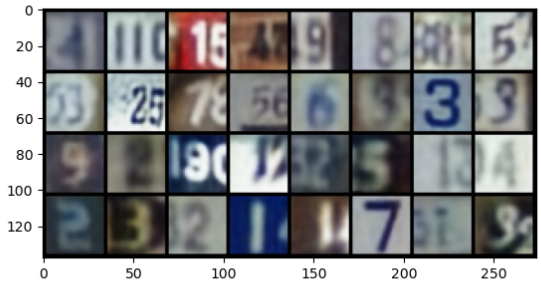


Fig. 19. The reconstructed images generated by DCVAE with 1024 latent features

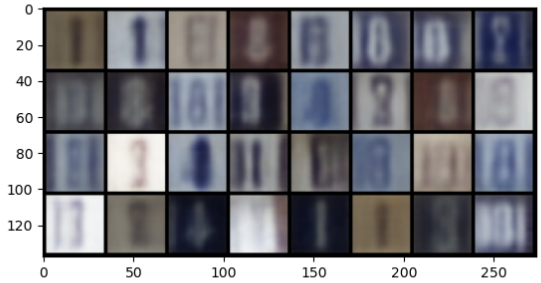


Fig. 20. The reconstructed images generated by DCVAE with 10 latent features

A. Vanila VAE

I start my experiment from a samll and simple VAE. The encoder is only one linear layer with ReLU function and the decoder layer is a two linear-layer model. I trained it for 10 epochs until it converges and the reconstructed images is shown as figure 18. The reconstructed images are very fuzzy which means only linear layers with simple non-linear transformation is not enough

B. DC-VAE

Since, we have discussed the success of convolution layers. Then I use Convolution layers to generate the VAE. The encoder is a conv layer models with leakyReLU activation. The devoder is also a four conv layer (ConvTranspose2d) model with ReLU activation. I trained the model for 100 epochs. The reconstructed images are shown as figure 19. The result is satisfying.

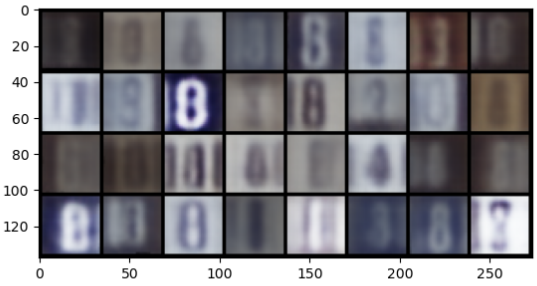
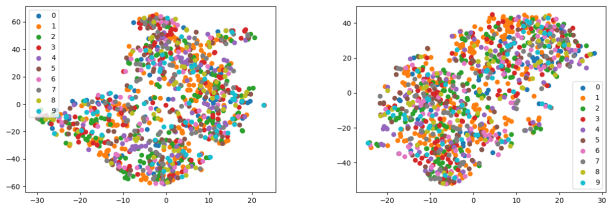


Fig. 21. The reconstructed images generated by DCVAE with 100 latent features



(a) 1024 (b) 10

Fig. 22. The TSEN visualization of latent features

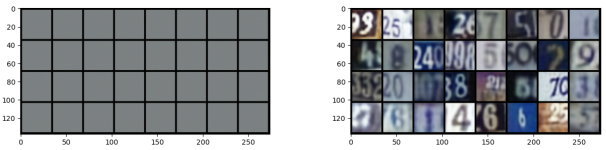
C. Latent Explanation

However, set a 1024 latent vector is lack of explanation. The only intution behind that is the latent dimension of CNN is 1024. But we are doing a 10-class classification with 10 numbers. As a result, I design an experiment to check whether VAE is learning human knowledge of latent vectors. I then change the latent dimension into 10 and the reconstructed result is shown as figure 20 and the result shows that the model lay some attention on unrelated information such as the color and styles of the background instead of the numbers.

Then I change the latent vectors into 100 dimension and the result is shown as figure 21. The model still fails to capture critical information. So, I wonder if it is true that the 1024 latent vectors capture the important informatoin or just it project the image into a better space. I use tSNE to visualize the 1024 and 10 latent features. As shown in figure 22, both have bad discriminatons of features which means the 1024-DCVAE just project the images to a better space.

In [18], the author rewrite the coding of svhn into (y, z) where $z \sim \mathcal{N}(0, I)$ and $y \sim (0.1, \cdots, 0.1)$ and the model is able to generate different model with different appointed backgrounds. But actaully, VAE has no ability to do that since all latent vectors are follow the normal distribution. The reason to do such entaglement is still because features are porjected to a better space through optimization.

As a result, we need a particular method to control how VAE concentrates the attention on useful information.



(a) No part1 (b) No part2

Fig. 23. The reconstructed images generated by DCVAE without part1 loss and without part2 loss

D. Loss Impact

As we have discussed that the two parts of the loss functions have different meaning. I try to isolate two parts and analysis how VAE works. The figure 23 shows the reconstructed images with isolated losses. The images generated without part2 is still clear but the model has no ability to adapt to unseen vectors. This means that if we do not have dreams, we still can get good realities but we do not have the potential to explore the unknown worlds. The images generated without part1 is failed since only dream is not enough to reduce the gap between the reality. As a result, we also need the guidence and link the dream and the reality. In [19], the authors give out detailed analysis about how to dientagle the loss functions and express high dimension data.

As a result, we need a better explainable VAE to further investigate the problem. In our project2, we try to disentagle the loss functions and give a better interpretention about how good loss function can instruct the VAE and its explanation ability. In project2, our group give a very detailed and deep insight about VAE.

IX. Code Intergrity

Most of the code are based on pytorch framework for better GPU computation. The loader of dataset is through *torchvision.SVHN*. All basic logistic regression models are wrote by myself but the optimization through SGD is implemented by *torch.SGD*. The loss functions are handwrote. The SVM classfier is implemented through *sklearn.svm.SVC*. LDA and GMM is wrote by myself except I use *torch.eig* to get eigenvectors. For neural networks, the model structures are all implemented by myself. The LSTM cell and CNN layers are from *torch.nn.RNN, LSTM, Conv2D, BatchNorm2d, Linear, ReLU, Tanh, MaxPool2d, AvgPool2d*.

About visualization part, I use the PCA and TSEN in sklearn. The guided backpropagation is implemented by myself. The GradCAM is through python pytorch_grad_cam package.

In OC, I only submit the code because of file limitation. During all my experiments, I have done the detailed loggings including the scripts (codes) run under that experiment, the settings of hyper parameters, the changing of loss and accuracies and also the best model parameters. If there is any question, please ask me for these files.

The math parts in the report are all based on professor's materials and myself if no particular reference is stated. Figure 9 and figure 14 is from the internet.

X. Conclusion

In this project, through implementing various kinds of machin learning models, I have deep understanding of basic machine learning concepts. During the project, I also discover some interesting pheomenons and design small experiments to prove mY ideas. After checking these hypotheses, I also deduct some my own understanding and conclusion. All in all, I got great fun.

Acknowledgement

I give MY sincerest appreciation to Prof. Zhang, Dr. Cheng and Dr. Ren for giving me this chance to contribute my efforts in the work of machine learning. I also thank them for their generous help.

References

[1] S. S. Keerthi and C.-J. Lin, "Asymptotic behaviors of support vector machines with gaussian kernel," *Neural computation*, vol. 15, no. 7, pp. 1667–1689, 2003.

[2] H.-T. Lin and C.-J. Lin, "A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods," *submitted to Neural Computation*, vol. 3, no. 1-32, p. 16, 2003.

[3] A. BenHur and J. Weston, "A users guide to support vector machines," in *Data mining techniques for the life sciences*. Springer, 2010, pp. 223–239.

[4] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1314–1324.

[5] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[6] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[7] J. Ren, M. Li, Z. Liu, and Q. Zhang, "Interpreting and disentangling feature components of various complexity from dnns," *arXiv preprint arXiv:2006.15920*, 2020.

[8] R. Kemker and C. Kanan, "Self-taught feature learning for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 5, pp. 2693–2705, 2017.

[9] Q. Zhang, Y. Yang, Q. Yu, and Y. N. Wu, "Network transplanting," *arXiv preprint arXiv:1804.10272*, 2018.

[10] Q. Zhang, Y. N. Wu, and S.-C. Zhu, "Interpretable convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8827–8836.

[11] D. Alvarez-Melis and T. S. Jaakkola, "Tree-structured decoding with doubly-recurrent neural networks," 2016.

[12] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *arXiv preprint arXiv:1508.01991*, 2015.

[13] H. Mhaskar, Q. Liao, and T. Poggio, "Learning functions: when is deep better than shallow," *arXiv preprint arXiv:1603.00988*, 2016.

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[15] E. Salinas and L. F. Abbott, "A model of multiplicative neural responses in parietal cortex," *Proceedings of the national academy of sciences*, vol. 93, no. 21, pp. 11 956–11 961, 1996.

[16] R. L. Hahnloser, "On the piecewise analysis of networks of linear threshold neurons," *Neural Networks*, vol. 11, no. 4, pp. 691–697, 1998.

[17] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv preprint arXiv:1504.00941*, 2015.

[18] R. Shu, S. Zhao, and M. J. Kochenderfer, "Rethinking style and content disentanglement in variational autoencoders," 2018.

[19] B. Esmaeili, H. Wu, S. Jain, A. Bozkurt, N. Siddharth, B. Paige, D. H. Brooks, J. Dy, and J.-W. Meent, "Structured disentangled representations," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 2525–2534.