Denoising and Classifying with Auto Encoder
— Group2 Final Report

1. Introduction

1.1 Denoising

Due to the imperfections of imaging systems, transmission media and recording equipments, digital images are often interfered by various noises during their formation, transmission and recording processes which may affect the visual effects of images and even hinder people's normal recognition. Thus, image denoising, which is the process of reducing noise in digital images, is essential in various fields.

The denoising part of this project aims to eliminate the noise of the given images by using auto encoder. Auto encoder is an unsupervised model that contains two parts, the encoder and the decoder. The encoder encodes the original representation into a hidden layer representation, while the decoder decodes a hidden layer representation into the original representation. The training goal is to minimize the reconstruction error. The dimension of the hidden feature is generally lower than the dimension of the original feature.

The encoder and the decoder will be implemented with CNN. The image denoising task will be completed by learning the mapping from the noisy picture to the original picture.

1.2 Classification

After successfully eliminating the noise of each image in the denoising part, we next aim to recognize the object on each image. Convolution kernels of convolution neural network can be used to detect the vertical edge images, relative to the entire image as a vector of the connection layer, it ignores the image itself of the two-dimensional space characteristics, convolution operation is very good at handling this kind of local feature.It can more effectively extract more useful information in images (as in the edge of the image). In this paper, the most classic CNN algorithms VGG-16, Alex Net and Res Net-18 are selected to complete the image classification task.

2. Research Methods

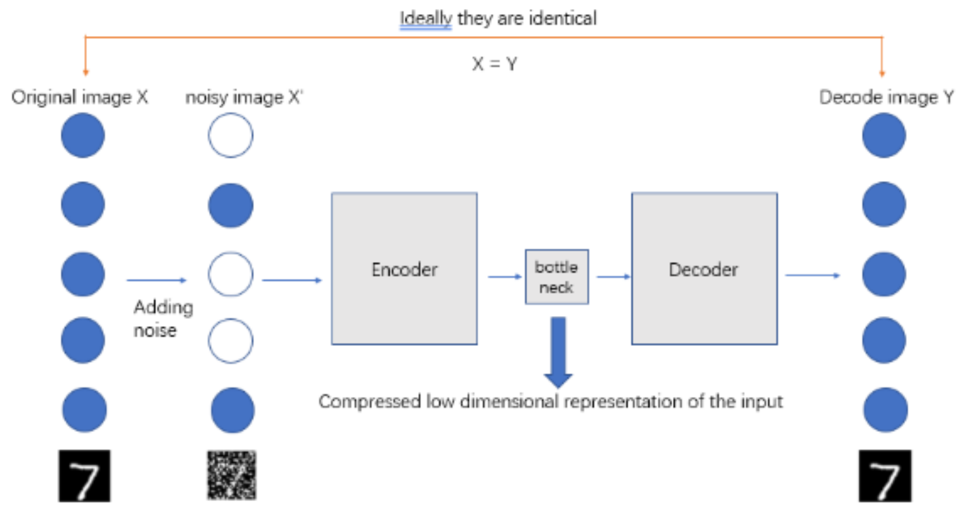


Fig.1 Denoising Flow

We use MNIST (Modified National Institute of Standards and Technology dataset) as our dataset. MNIST is the handwritten digit recognition dataset which contains 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9. The MNIST dataset is from the National Institute of Standards and Technology (NIST). The training set consists of handwritten numbers from 250 different people, 50 percent high school students and 50 percent people who work at the Census Bureau. The test set is the same proportion of handwritten numeric data.

It could be imported from Keras by entering command:

```
from keras.datasets import mnist
```

The dataset is available in http://yann.lecun.com/exdb/mnist/. It contains 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images.

We first load the MNIST dataset. Then we add noise to each image by randomly picking pixel points from each image and randomly adding those pixel points to each image. Now, an MNIST dataset with noisy images are prepared (Fig.2). The whole denoising process is shown in Fig.1.

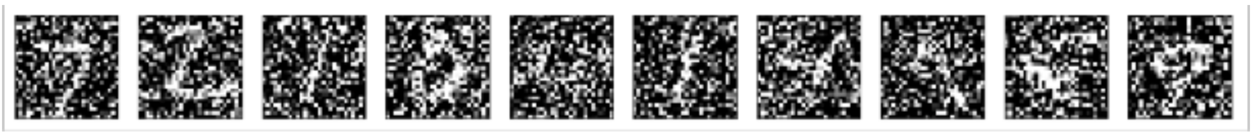


Fig.2 Images after adding noise (10 examples)

We then use three models, VGG-16, Alex Net, and Res Net-18 to eliminate the noise of each image and recognize the hand written numbers on each image.

2.1 VGG-16

2.1.1 Introduction

VGG-16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford. It makes the improvement over Alex Net by replacing large kernel-sized filters with multiple 3×3 kernel-sized filters one after another.

VGG-16 contains a stack of convolutional layers, where the filters were used with a small receptive field whose size is 3*3. It also utilizes 1*1 convolution filters, which can be seen as a linear transformation of the input channels. The convolution stride is fixed to 1 pixel. The spatial padding of convolutional layer input is such that the spatial resolution is preserved after convolution. So the padding is one pixel for 3*3 convolutional layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the convolutional layers. Max-pooling is performed over a 2*2 pixel window with stride 2.

2.1.1.1 Why VGG-16 but not VGG-19?

VGG-16 has 16 layers, but VGG-19 has 19 layers with extra convolution layers in the last three blocks, which indicates that VGG-19 may have a better performance than VGG-16. However, VGG-19 requires more memory than VGG-16. Therefore, in this project, I still choose VGG-16 but not VGG-19 since this is only a simple denoising and classifying task so both of them can have good performance, and there is no need to use up a lot of memory just to get a slightly better result.

2.1.2 Denoising

We first let each input image, who has a size of 28*28*1, pass through a convolutional layer with 32 filters and a kernel size of 3*3, then we get an output image whose size is 28*28*32. Next, we let this output image pass through another convolutional layer with 64 filters and a kernel size of 3*3, resulting in an output image of size 28*28*64. Then we let each output image pass through a 2*2 max pooling layer, giving us a 14*14*64 output image. Next, we let each output image pass through two 3*3*128 convolutional layers and one 2*2 max pooling layer, resulting in an output image of size 7*7*128. Next, we let each output image pass through three 3*3*256 convolutional layers and one 2*2 max pooling layer, resulting in an output image of size 4*4*256. Then, we let each output image pass through three 3*3*512 convolutional layers and one 2*1 max pooling layer, resulting in an output image of size 2*4*512. Then we again, let each output image pass through three 3*3*512 convolutional layers and one 2*1 max pooling layer. Finally, each output image has a size of 1*4*512. This is the end of the encoding process. The encoding flow is shown in Fig.3. Notice that our original input image size is 28*28*1, but our current output image size is 1*4*512. So we still need a decoding process to make the image size back to 28*28*1.

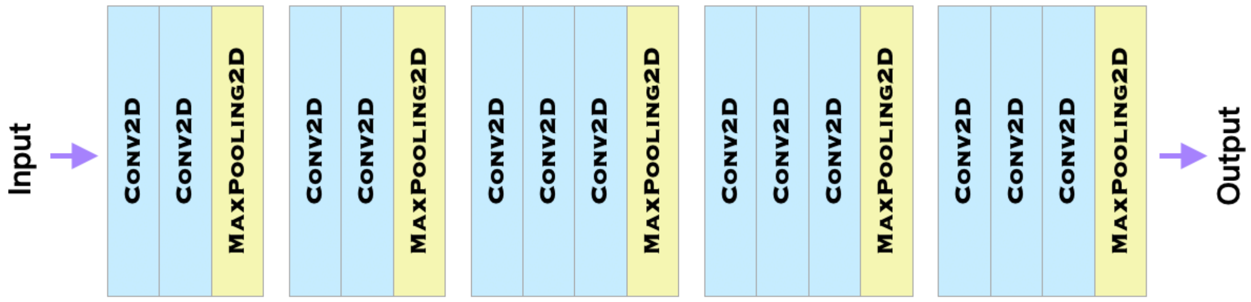


Fig.3 VG-16 Encoding Flow

The decoding process is just the inverse of the encoding process. We still let each image pass through those VGG layers, but in the inverse way. We first let each output image we have got from the encoding process (current size is 1*4*512) pass through three 3*3*512 convolutional layers and one 2*2 max pooling layer twice. Then, let each of those output images pass through three 3*3*256 convolutional layers and one 2*2 max pooling layer, followed by two 3*3*128 convolutional layers and one 2*2 max pooling layer. Finally a 3*3*64 convolutional layer, a 3*3*32 convolutional layer, an 1*1 max pooling layer, and a 3*3*1 convolutional layer. So now, our image size goes back to 28*28*1. The flow of the decoding process is shown in Fig.4.

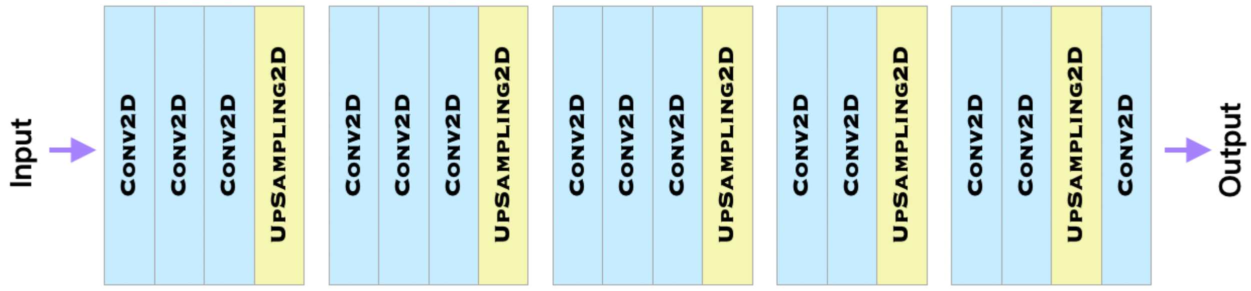


Fig.4 VGG-16 Decoding Flow

After the encoding and decoding processes, we train the model for 100 epochs with a batch size of 32. Then we randomly pick 10 images and print out the denoising results to visualize how the model performs. However, the run time of each epoch is a bit long (about 16s each), and the denoising results are not very satisfying. Therefore, we then set our batch size to be 64 and 128, and notice that when the batch size is 128, we get better denoising results (Fig.) as well as a short running time (8s for each epoch).

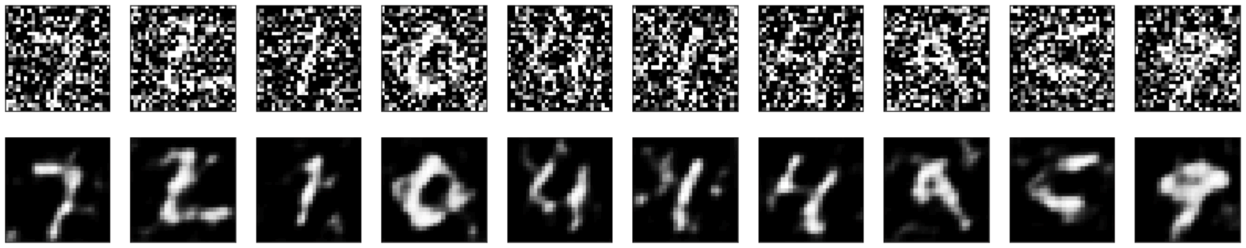


Fig.5 VGG-16 Denoising Results (10 examples) Batch_size = 32

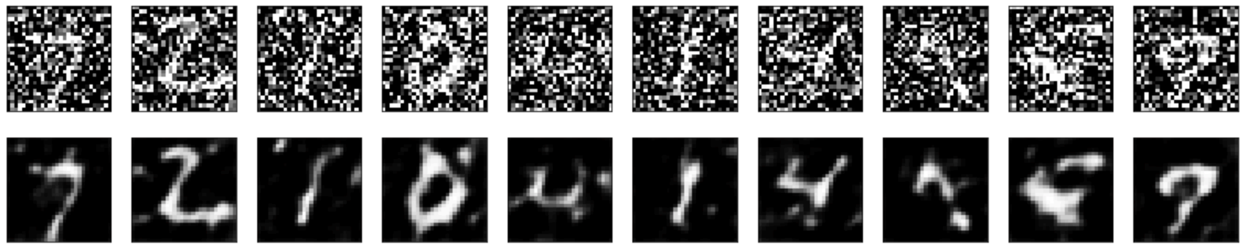


Fig.6 VGG-16 Denoising Results (10 examples) Batch_size = 128

2.1.3 Classification

After successfully eliminating the noise in the denoising part, we aim to recognize the hand written numbers. We first save the clear images we have got from the denoising part. Then we still let each image go through all the layers of VGG-16. We define a model which contains all the layers of VGG-16.

Next, we evaluate our model using k-fold cross-validation. We choose our hyper parameter k to be 5 in order to provide a baseline for both repeated evaluation and to not be so large as to require a long running time.

We train the model for 10 epochs with a default batch size of 32. The test set for each fold will be used to evaluate the model during each epoch of the training run, so that we can later create learning curves.

2.2 Alex Net

2.2.1 Introduction

Alex Net was created by the doctoral student of Geoffrey Hinton and won the 2012 ImageNet competition. It is really a milestone in deep convolutional neural networks. Before Alex net was invented in 2012, this field had been silent for a very long time. The only usable convolutional neural network before the invention of Alex Net is LeNet-5 created in 1986.

Compared with Lenet5, this model has 6 major contributions: 1. introduced the dropout method in the Linear Layer to prevent overfitting. 2. introduced ReLU as the activation function instead of the Sigmoid Function used in LeNet-5 to prevent the gradient disappear. 3. introduced the Data Augmentation method to help us get more data and thus prevent the overfitting. If our training data is very scare, we could use data augmentation to increase the size of our data thus preventing overfitting. 4.using CUDA to train the model to speed up the running time. From figure 2.2.1, we can see the author distribute the total parameters into two GTX580 GPU because the ram capacity of GTX580 is not enough. 5. Using Max pooling as the pooling layer instead of the average pooling because the blurring effect of average pooling. 6. Introduced LRN layer to enhance the generalization ability of model.

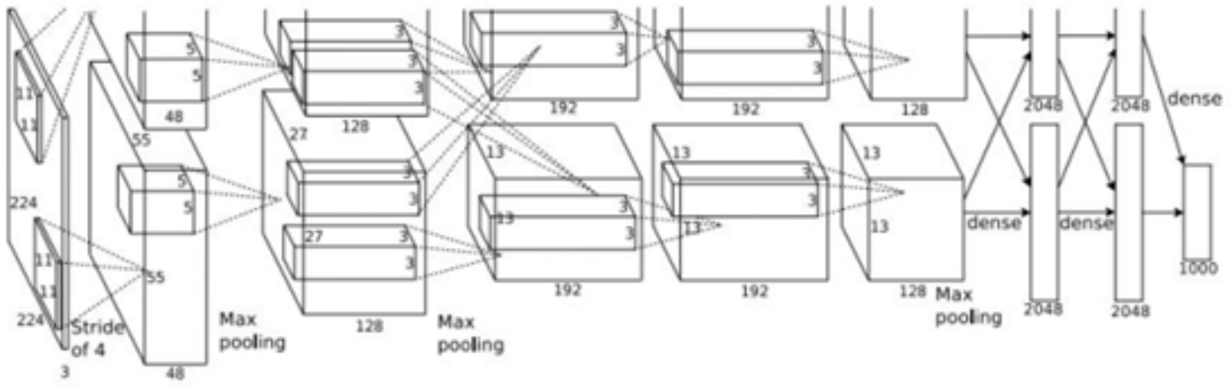


Fig.7 the Alex Net model

2.2.2 Denoising

Unlike the convolutional layer in VGG which doesn't affect the input size. We can see the first convolutional layer of Alex Net significantly decrease the size of input. Therefore, image resize from 28 to 224 become an indispensable part for us in this task. In this denoising task, we only used the convolutional layer in Alex Net to extract features because we don't need the Linear model to do prediction. Also, since the data in MNIST dataset is very big, we don't need to do data augmentation any more to increase our input data.

The encoding part for this denoising task is very straight forward, we just put our data into the convolutional part of Alex Net to extract features and decreased the size into a bottleneck. Both the first convolutional layer and 3 Max pooling layer would help to decrease the size. Finally, we could get an output with the size of 7*7.

The decoder part is just the inverse implementation of the encoder part. How do we realize this implementation? Since Alex Net contain convolutional layer to decrease the size of data, here we used the transpose of convolutional layer to realize this inversion of convolutional layer. We also apply Up sampling to inverse the process of max pooling. After the series of inverse process, we could happily get an output with the size equal to our original data.

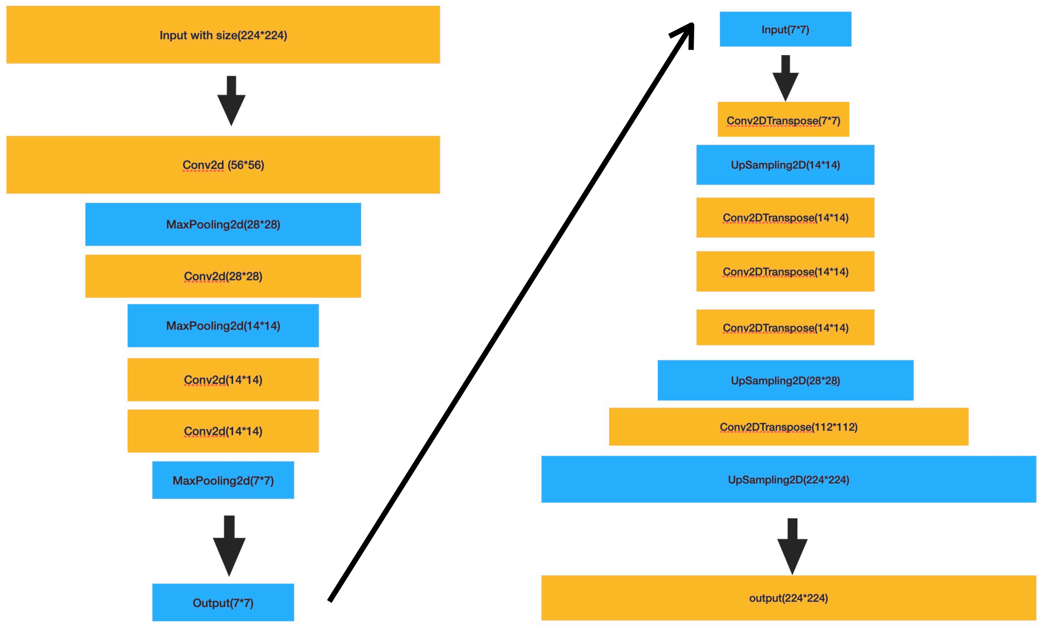


Fig.8 Alex Net - the main procedure for the denoising task

2.2.3 Classification

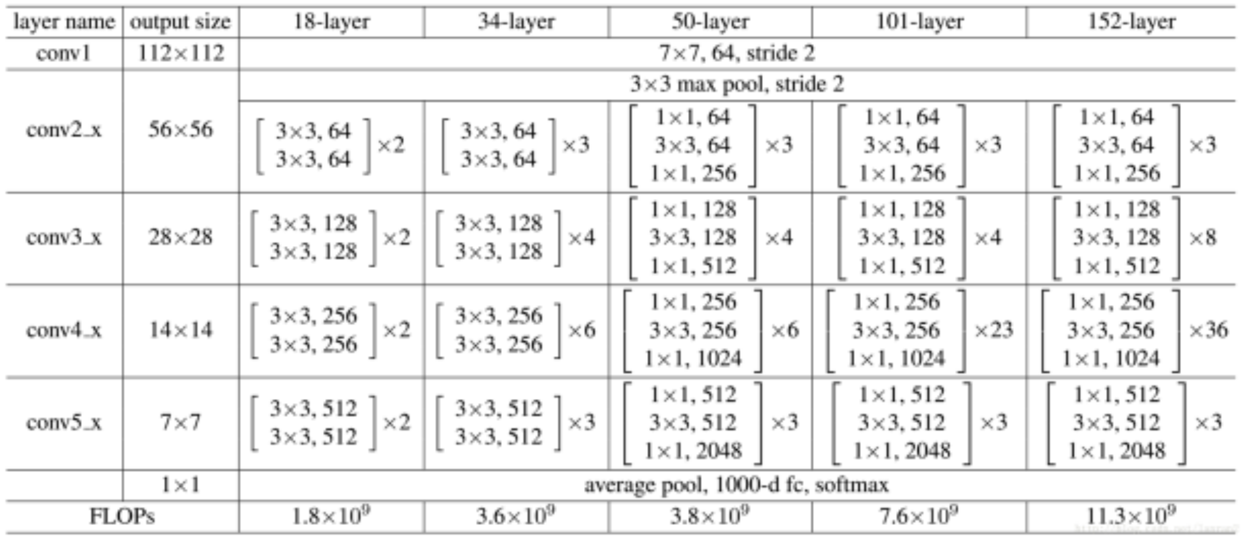
Since we would like to implement a “One-Stop” program, that is to automatically denoising the noisy image and then return the label of this image. We decided to build another model for classification.

Here We use the full Alex Net model including the Linear model to help us do classifications. The Linear model contains two fully connected layers combined with the soft max function to get the highest probability label. We also apply dropout in the fully connected layers to randomly ignore some perceptron to prevent the overfitting and enhance the generalization ability of the model.

2.3 Res Net-18

2.3.1 Introduction

According to different network layers, ResNet is divided into 18,34, 50, 101 and 152. ResNet18 and 34 are mainly shallow feature extraction. Considering the size of MNIST data set is not so large, ResNet18 is selected for simulation in this paper.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fig.9 ResNet-18 Layers

The network structure of the model mainly includes five parts: Conv1, Conv2_X, Conv3_X, Conv4_X, conv5_X; Conv1 comprises a 7*7 convolution, Bn1 and Maxpooling layer. The 2-4 layers has similar structures and are all composed of two blocks, each contains two 3*3 convolution kernels. The biggest difference between VGG algorithm and itself is the introduction of residual learning, which establishes a shortcut connection between input and output, and can effectively solve the problem of network degradation with increasing depth.

2.3.2 Denosing

In this paper, the denoising algorithm is a self-encoder model, which is composed of encoding and decoding, and the network structure of each part is realized by imitating ResNet18. In the coding part, the convolution layer and pooling layer are used to convert the input 28*28*1 into 1*1*512, which is the hidden layer of the image. In this part, I fine-tune some parameters(such as step size, stride or padding) to ensure it works for our data set. The network structure of the decoded part is shown in the figure below.

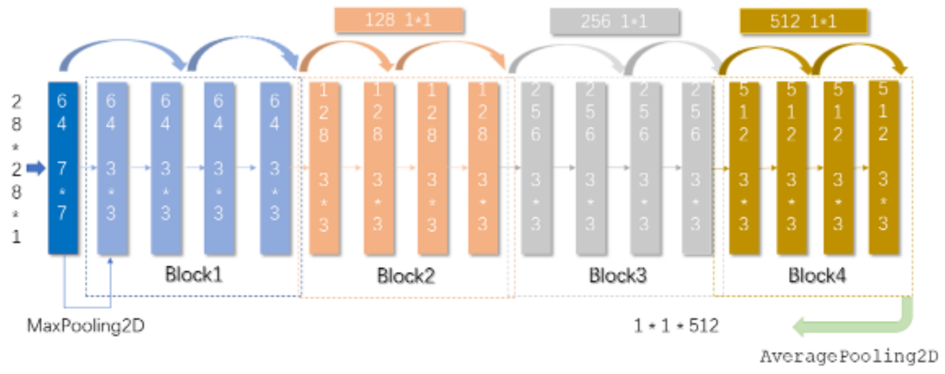


Fig.10 ResNet-18 Encoding Process

The decoding part (Fig.11) is the inverse process of the coding part. I use the method of upsampling to expand the selection of features. At the same time, in order to ensure that the size of input and output is consistent, I use a 1*5 convolution kernel for the last layer.

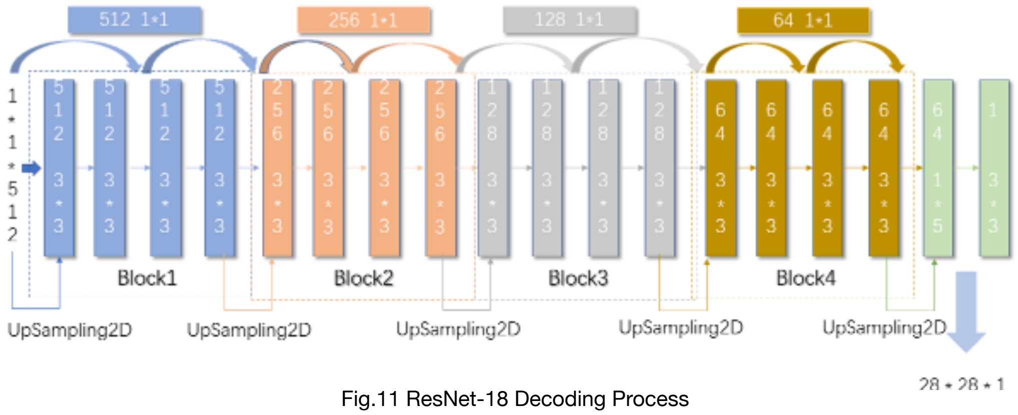


Fig.11 ResNet-18 Decoding Process

2.3.3 Classification

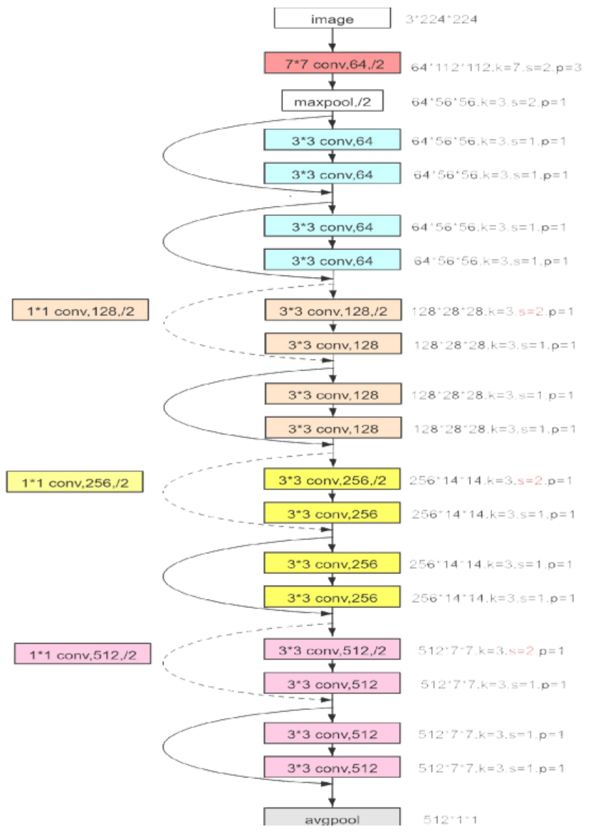


Fig.12 ResNet-18 Network

My classification part of the network structure and coding layer are roughly similar, all adapted on the basis of ResNet-18. It adopts four layers of Resnetblock. Considering the overfitting problem, the last block is removed in this paper, so that the image size after the convolution layer is 4*4*256. Finally, through a Flatten and full connection layer, the network model is built (slightly different with the figure).

3.Outcomes and Evaluation

3.1 VGG-16

3.1.1 Denoising Results

10 examples of the denoising results are shown in Fig.13.

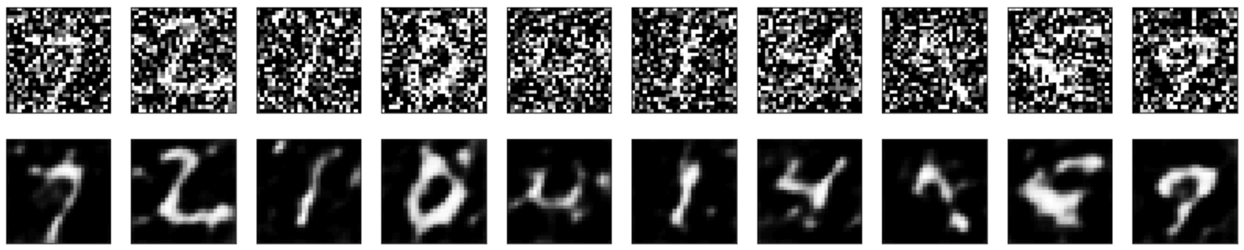


Fig.13 VGG-16 Denoising Results (10 examples)

3.1.2 Classification Results

A classification example is shown in Fig.14.

INPUT:

OUTPUT:

```
The number on the image is  2
```

Fig.14 VGG-16 Classification Result (1 example)

We used the 5-fold cross-validation to evaluation the model. An example of the five accuracies are shown in Fig.15, and the learning curve is shown in Fig.16. We can see that the accuracies are all above 99%. Moreover, the cross entropy loss is decreasing and the accuracy is increasing, indicating that the model is good.

```
> 99.042
> 99.133
> 99.083
> 99.342
> 99.117
```

Fig.15 VGG-16 Accuracies (1 example)

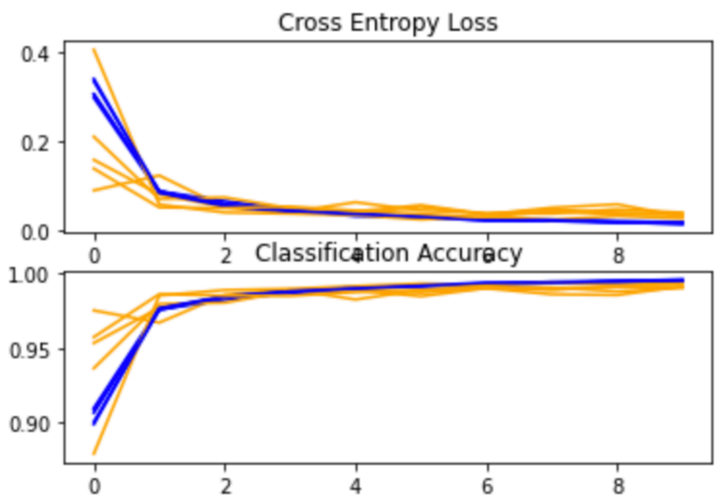


Fig.16 VGG-16 Learning Curves

3.2 Alex Net

3.2.1 Denoising Results

I used adadelta as the optimizer and binary-entropy as the loss-function and trained this model for 100 epochs. Here is the leaning curve we got from training:

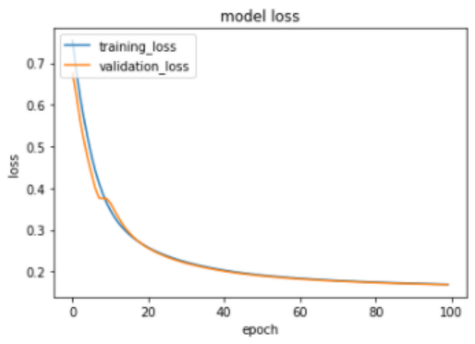


Fig.17 AlexNet - Denoising Model Loss

We can see the loss is constantly decreasing both in training data and validation data, and there are no overfittings or underfittings occur. Here is the denoising result example:

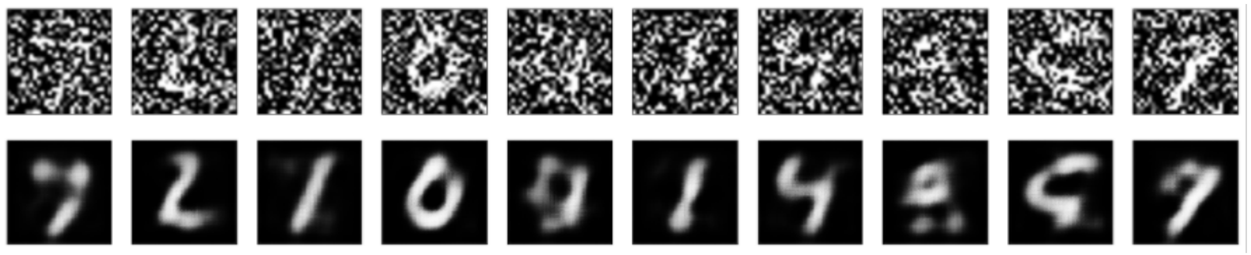


Fig.18 AlexNet - Denoising Results (10 Examples)

we can see the decoded images we got are more clear than the decoded images got from the other two models.

3.2.2 Classification Results

I used adam as the optimizer and binary-entropy as the loss-function and trained this model for 50 epochs. Here is the learning curve we got:

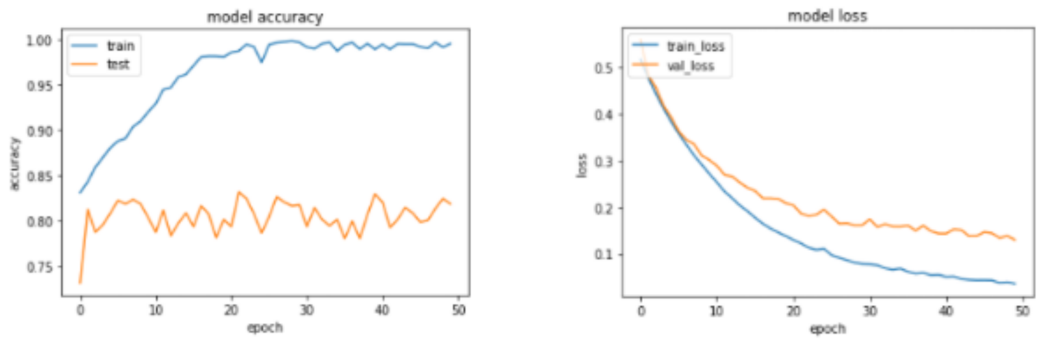
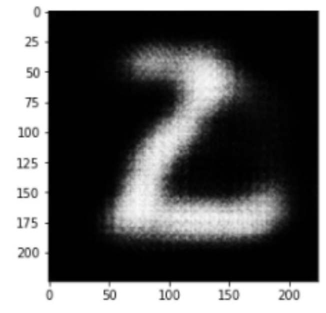


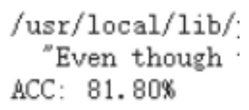
Fig.19 AlexNet Model Accuracy and Model Loss

However, we can see the model's performance on training data is significantly better than on validation data, indicating the overfitting. Here is the classification result example:



```
The prediction result is 2
```

Fig.20 AlexNet Prediction



```
/usr/local/lib/:
  "Even though
ACC: 81.80%
```

Fig.21 AlexNet Accuracy

I used implemented evaluation function in KERAS, (Not using k-fold validation because the size is 224 and the running time could be very long). The accuracy is not very good, only 81. I think this is because the ram capacity of my computer is not enough, I only used part of the whole MNIST data set as my training data thus cased overfitting.

3.3 Res Net-18

3.3.1 Denoising Results

During the training of the autoencoder, I set the ratio of the training set to the validation set to 6: 1. The optimizer is set as adadelta, the batch_size input for each round is 128, and the data is mixed before each round of iteration. Considering the complex network model and performance problems of hardware equipment, the model is trained 50 times in this paper. The loss value and denoising effect after training are shown in the figure below:

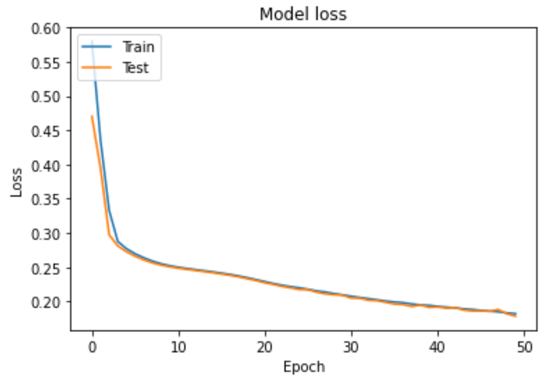


Fig.22 Res Net-18 Model Loss

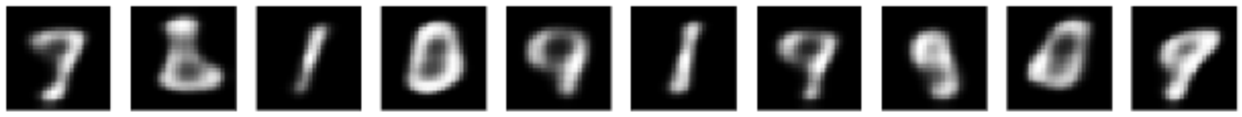


Fig.23 Res Net-18 Denoising Results (10 examples)

3.3.2 Classification Results

Data preprocessing:

1. load and save the denoised image data set as the test set of classification network structure
2. The label data is coded one-hot to make it conform to the network input specification

In this paper, MNIST data set is first used to train its own network structure. The data ratio between training set and verification set is about 9:1, the optimizer selects adadelta, the loss function is binary_Crossentropy, and the ratio between verification set and test set is 3:5. The effect of model training for 100 times is as follows:

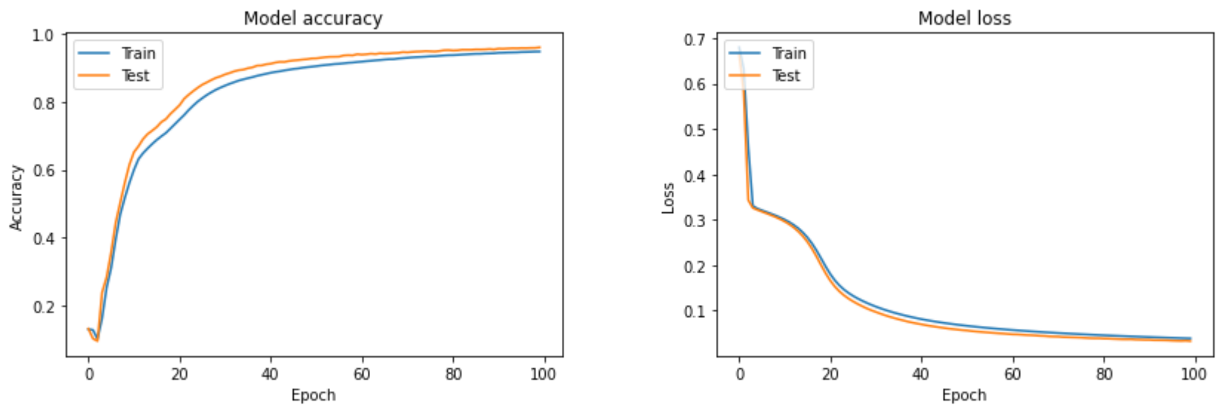


Fig.24 ResNet-18 Model Accuracy and Model Loss

According to the analysis of the result graph, in the course of 100 training times, the accuracy of the training set is increasing, and the loss value is decreasing. The accuracy of the test set is constantly rising, and the loss value is constantly decreasing. There is an approximate monotonic relationship between the two, indicating that the model is in constant learning, and there is no fitting phenomenon, and the fitting degree for the task is good.

4.Conclusion

We first compare the denoising results of the three models (Table.1).

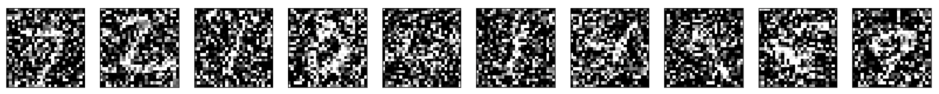
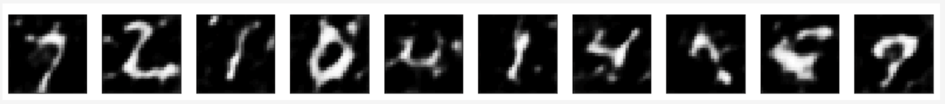
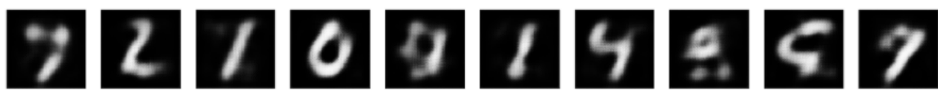
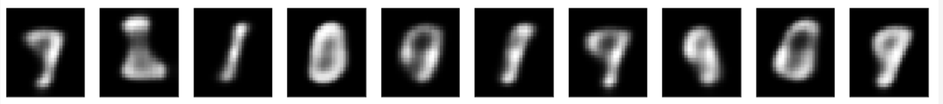
Noisy images (10 examples)	
VGG–16	
Alex Net	
Res Net	

Table.1 Denoising Results of Three Models

By observing the table above, we notice that the method using Alex Net returns the clearest images.

Next, we compare the accuracies of the classification of the three models (Table.2).

Model	Average Accuracy
VGG-16	99.08%
Alex Net	89.00%
Res Net-18	99.29%

Table.2 Classification Accuracies of Three Models

By Table.2, we notice that Res Net-18 gives us the highest accuracy.

Thus, we conclude that if we want to get the best results for both denoising and classification, then we can use Alex Net for denoising and Res Net-18 for classification. If we only want to use one model for both denoising and classification, then we can choose VGG-16. Although it cannot give us the best result for both denoising and classification, it does not give us the worst result for both parts.

5. Reference

1. Building Autoencoders in Keras https://blog.keras.io/building-autoencoders-in-keras.html
2. Neurohive, Popular networks, VGG16 – Convolutional Network for Classification and Detection. https://neurohive.io/en/popular-networks/vgg16/
3. Jason Brownlee, “How to Develop a CNN for MNIST Handwritten Digit Classification” (2019). Deep Learning for Computer Vision. https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/
4. Shu, Mengying, "Deep learning for image classification on very small datasets using transfer learning" (2019). *Creative Components*. 345. https://lib.dr.iastate.edu/creativecomponents/345
5. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84–90. https://doi.org/10.1145/3065386
6. Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.03385
7. Keras Chinese official document. https://keras.io/zh/