EE115A: Research Project

Due on January 9, 2022

Professor Gao Fei

Yining Jiang | Yiwei Shu

2020531062 | 2020531063

ABSTRACT

Recently, emerging photoacoustic imaging technology (PA) of human brain is the most challenging clinical application in the domain of medical imaging. Up to now, there is still no transcranial human brain PA imaging results reported. Why is human brain imaging is so challenging to PA imaging? Because the thick skull scatters and attenuates the light and sound dramatically. In this passage, we will do a series background research of photoacoustic sonogram denoising, and then we will introduce our method based on U-net neural network and estimate its feasibility.

BACKGROUND RESEARCH

During the recent decades, there exist various methods in previous research studies of noise reduction, as Professor Gao mentioned in class, they can be divided into the following three categories depending on their principles:

1)The first one is based on electronic circuits:low-noise amplifier, filter circuit, lock-in amplifiers.etc. Underlying techniques of analogic circuits, they are designed so delicate. However, the previous studies we've learned about are more concentrated on using the circuit as a filter to remove noise from some sequential signals but image denoising. So we didn't pay much attention to the such specific circuit designing.

We did some research on lock-in amplifiers. The lock-in amplifying technique is a precise method to detect and recover the amplitude and the phase position of a given faint signal by using the reference signal. And the lock-in amplifier can extract the faint target signal from the drastic background noise, amplify and measure it. The following graph shows the basic structure of a lock-in amplifier:SR830.

The pivotal component is the Phase Sensitive Detection module(PSD), which can do the computation of the multiplication of the two signals. And by theoretical calculating, we get the output:

$$\begin{cases} X = \frac{1}{2}A_{amp}A_{in}A_r cos(\phi_s - \phi_r) \\ Y = \frac{1}{2}A_{amp}A_{in}A_r sin(\phi_s - \phi_r) \\ R = \frac{2\sqrt{X^2+Y^2}}{A_{amp}A_r} = A_{in} \\ \theta = \phi_s - \phi_r = \arctan(\frac{Y}{X}) \end{cases}$$

while A_{amp} is the DC gain, A_{in} is the input amplitude, A_r is the amplitude of the reference

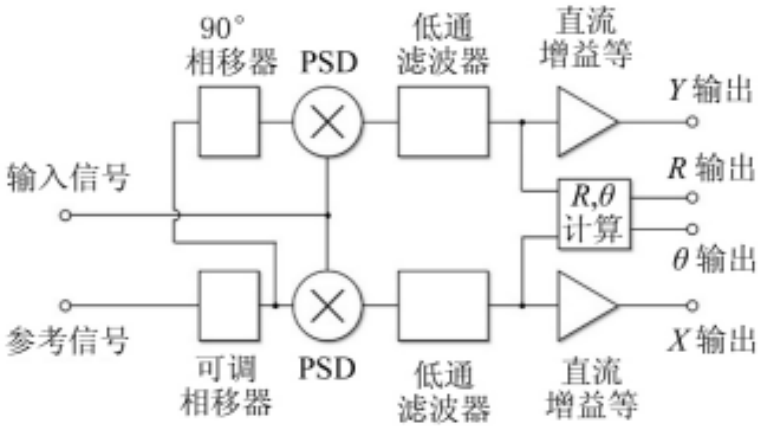

Figure 1: The structure plot of SR830

signal, ϕ_s is the phase of the target signal, ϕ_r means the phase of the reference signal.

We have an elementary idea of using SR830 to achieve our target of mage denoising:first, we can transfer the initial image into a one-dimensional signal, so does the reference image. After the process of filtering, we can get the result.

2)Another is to take signal processing ways, designing different filters with filter algorithm (low-pass, high-pass, band-pass, etc.) In this passage, we will demonstrate our try in designing filter algorithm and why it can't work effectively.

First, it is necessary to illustrate the Principal Component Analysis(PCA) algorithm, which is one of the most common algorithm for data dimensionality reduction. The reason why we take this method is that there may exist correlation between multi-variables, which may add great complexity to data analysis. Hence, PCA is a significant way which can remove the number of variables we need to analyze and minimize the information lost at the same time. The major steps of PCA can be described as: 1)Demeaning the sample. 2)finding the unit vector ω which can maximize the variance after dot mapping. At last, the task can be briefly described as: find ω which maximize

$$Var(X_{project}) = \frac{1}{m}\Sigma_{i=1}^{m}(X_1^{(i)}\omega_1 + X_2^{(i)}\omega_2 + ... + X_n^{(i)}\omega_n)^2$$

Second, our main idea is we can do the Fast Fourier Transform(FFT) to the image information, after that, apply PCA to the transform result, then we draw the spectrogram of the initial image. Similarly, same as the ground truth image. By minuting the ground truth information, we can get the spectrogram of noise. According to the features of the

spectrograms, we can design specific algorithm to remove the noises.

3)The other is using artificial intelligence ways, in this passage we take the method which is based on the convolutional neural network(CNN). Briefly, in order to train the CNN, we transfer the mission of image denoising into a learning task. Under the hypothesis that we have already got the clear images x without noises, we can train CNN by adding a step $n(x)$ that put noises into each photo. Then regard the neural network as a function $F_\Phi(x)$ with parameter Φ. After that, we can update the parameter in the network by minimizing 'error' $= \Sigma_i(x_i - F_\Phi(n(x_i)))$, that's the process of CNN denoising.

In our research, we chose the U-NET model as our basement. And here is the basic structure of U-net.

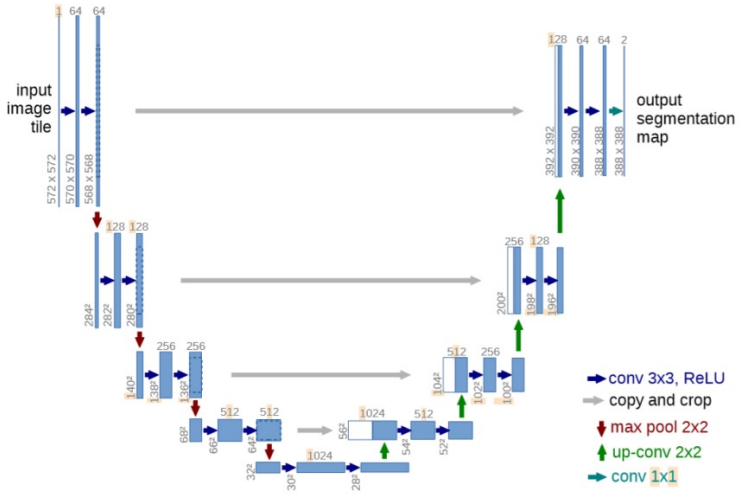

Figure 2: The structure of U-net

U-net is a kind of full convolutional neural network, while the left part in the graph shows its feature-extracting network: using convolution and pooling and the right part illustrates its feature fusion process: concatenate the plot from up sampling with the feature plot in the left part. While the up sampling can transfer the images with superior abstract features but in low resolution into ones with superior abstract features but in high resolution.

The main process may contain the following several parts: the convolutional layer, the pooling layer, the batchnorm layer the loss layer and adding regularization to the convolutional neural network. As we all know, convolution layer is defined as an operation that take

the inner product of the image and the filter matrix, which can be expressed as:

$$H_{out} = \left[\frac{H_{in} + 2 \times padding[0] - dilation[0] \times (kernal - size[0] - 1) - 1}{stride[0]} + 1\right]$$

$$W_{out} = \left[\frac{W_{in} + 2 \times padding[0] - dilation[0] \times (kernal - size[1] - 1) - 1}{stride[1]} + 1\right]$$

The purpose of convolution is extracting the characteristic of the input.

Pooling layer is designed to remove some unnecessary information in the feature map and compress the sample, reduce the number of parameters. In our research, we take the average-pooling, which means we take the average value of all values in a block to represent them.

As for the layer of BatchNorm, we carry out some normalizing means, normalize the distribution of the input to each layers into a standard normal distribution, which can extremely shrimp the absolute difference and increase the relative difference , i.e. accelerate the training.

Additionally, we also define a loss function to compare the input and output of the neural network, and train it by minimizing the loss function.

RESEARCH PROCESS

In the beginning, we started from filter algorithm designing–using filter to remove the noise in the graph. However, the truth is that the outcome is not quite satisfying because the spectrogram of the initial image and the noise signal are seemed to have striking similarity, which leads to bad denoising result. So we decided to use the neural network method based on U-NET to do the job, by effort, we drew significant outcome at last.

We will show our research result using filter at first. (Code has been attached in the appendix part). We define the noise N as $N = Y - X$, while Y is the matrix as the ground truth image and X is the matrix of signal-with-noise image. We apply Fast Fourier Transform to all samples, and then we get the following spectrograms.

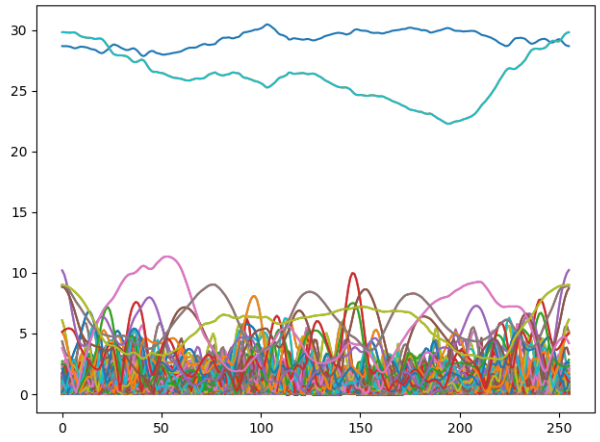

Figure 3: spectrogram of signal-with-noise

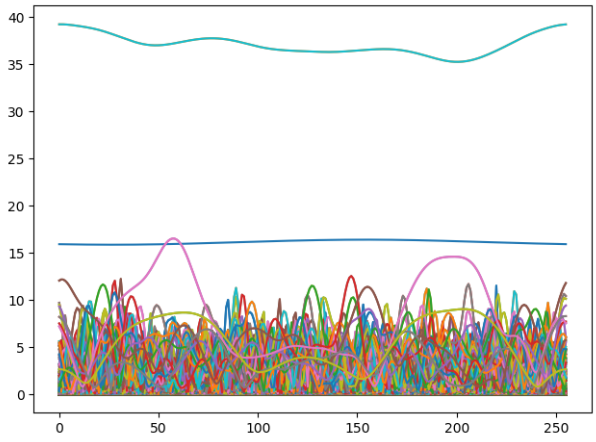

Figure 4: spectrogram of ground-truth

Apply PCA, we get,

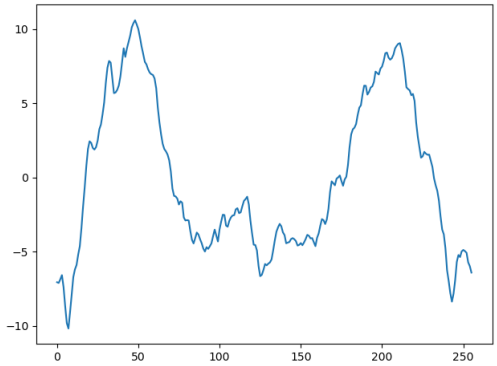

Figure 5: 1-dim spectrogram of signal-with-noise

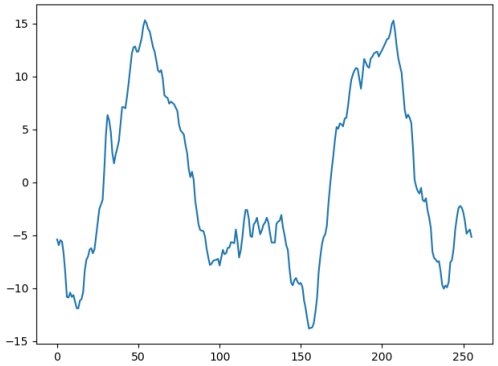

Figure 6: 1-dim spectrogram of ground-truth

By definition, we can also draw the plot of the noise signal:

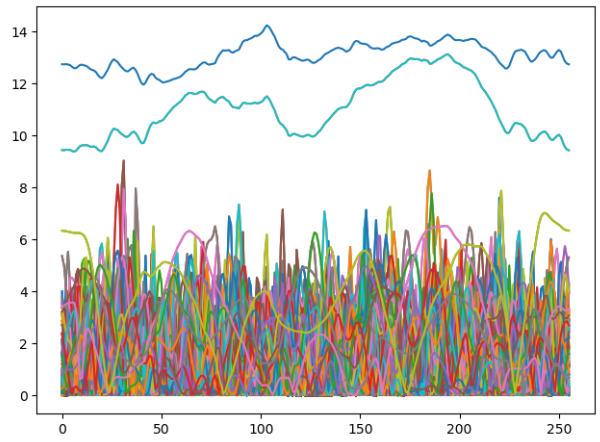

Figure 7: spectrogram of noise

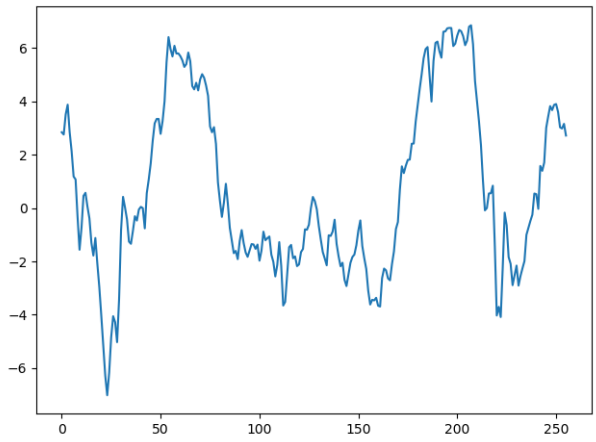

Figure 8: 1-dim spectrogram of noise

After observing the spectrogram, it seems that there exist striking correlation between the noise signal and the ground truth signal, hence, we are convinced that it's presumably too difficult to design the denoising algorithm and perhaps a disappointing result is unavoidable. Then our research direction turned to U-net structure.

As we mentioned in the background research, we use the mature U-net method to achieve the denoising target. There are several steps to describe our work. First, data-loading. And the corresponding code part is 'dataload.py'. This document is used to import the initial image and the ground truth image. They are shown as 256×1200 matrices. Second step is called data-setting and importing, which is achieved in 'dataset.py'. In this document, all the matrices are transformed into slices and concatenate together as a long chain, which can be read by our U-net model, and then are imported into our U-net model 'model.py'. We also use PSA algorithm here to realize the dimensional reduction. It has to be emphasized that in this step, the data in the chain are out-of-order, which is designed to prevent overfitting of the parameters. Third, train the model. It's worthy to demonstrate some meanings of the parameters. In the layer of batch norm, the parameter "batch size" means the number of the samples that are chosen in one single training. After experiment, we estimate 128 is a good batch size which can achieve a fabulous balance between the training quality and training time. As for the number of training "epoch", because of the time limit, we set epoch only at 50,000, but the outcome reached our satisfaction. After training, our model is

expected to be tested. Based on the initial signal-with-noise image, we design the data used for testing, which are stored in the document 'test-dataset.py'. The data for test is different from the ones for training, for the test data are traversal and in order. After observing the output images by eyes, then at last, we compute several indicators to analyze and estimate the feasibility of our research.(Code has been attached in appendix.)

RESULT DEMONSTRATION

The following illustrations show the 8 output images together with the 8 ground truth images.

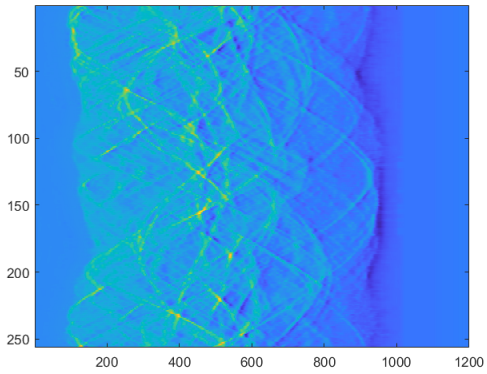

Figure 9: outcome 1

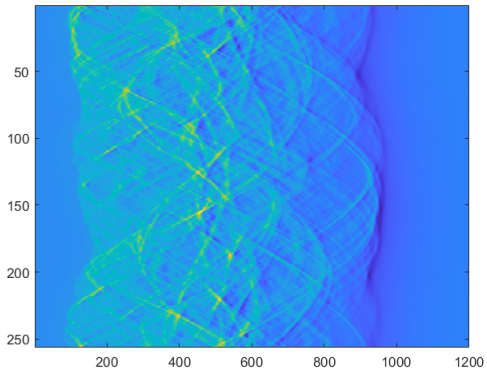

Figure 10: ground truth 1

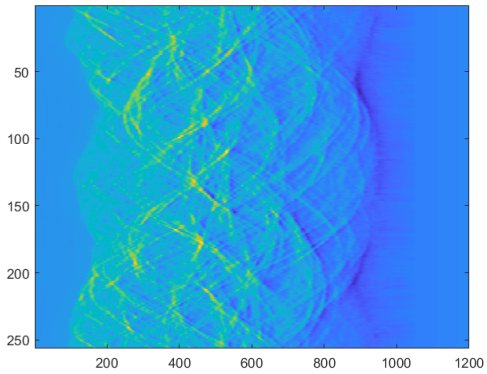

Figure 11: outcome 2

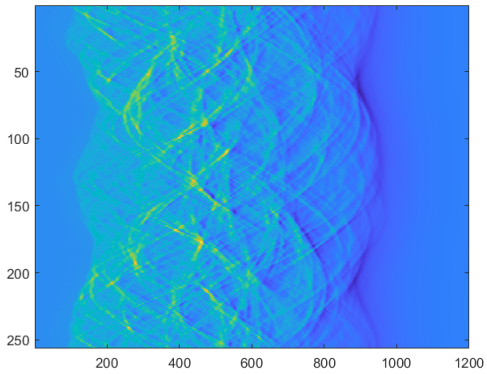

Figure 12: ground truth 2

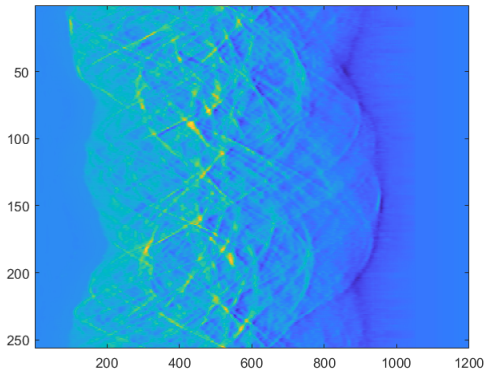

Figure 13: outcome 3

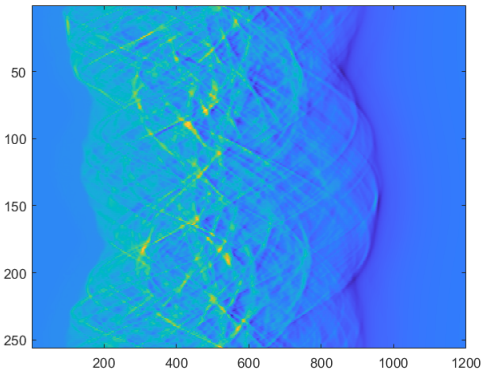

Figure 14: ground truth 3

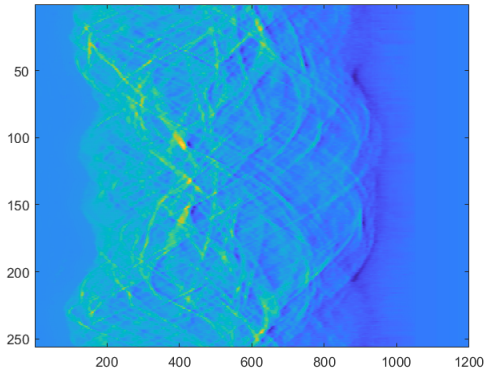

Figure 15: outcome 4

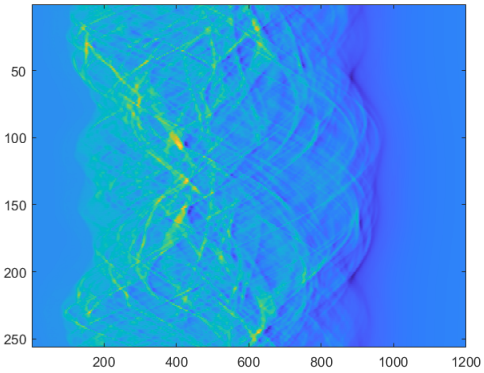

Figure 16: ground truth 4

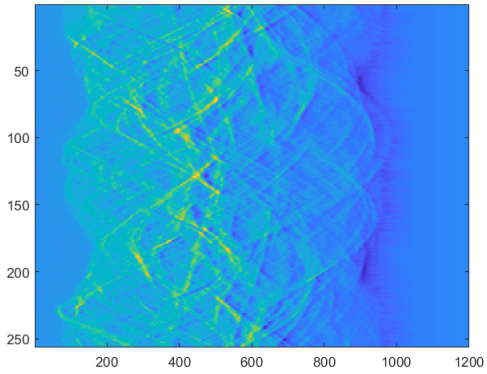

Figure 17: outcome 5

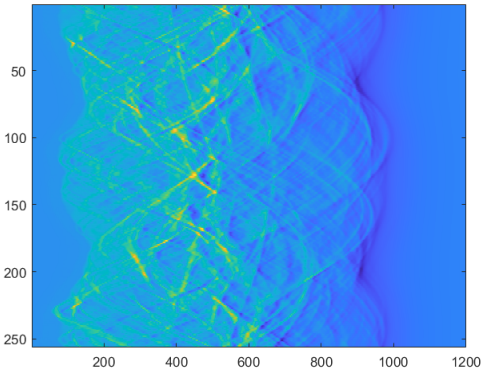

Figure 18: ground truth 5

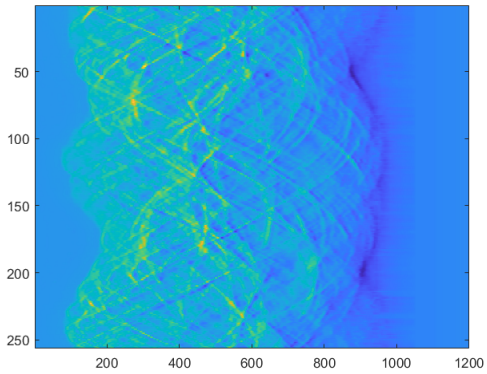

Figure 19: outcome 6

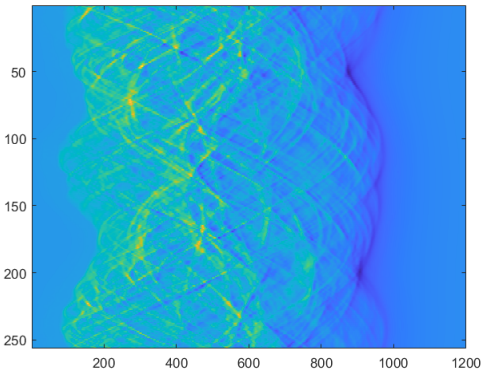

Figure 20: ground truth 6

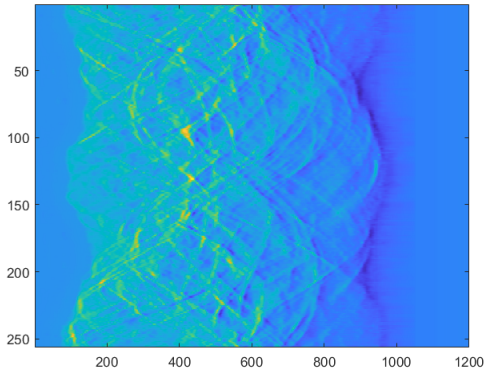

Figure 21: outcome 7

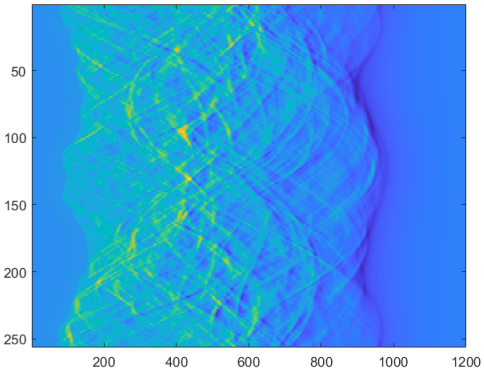

Figure 22: ground truth 7

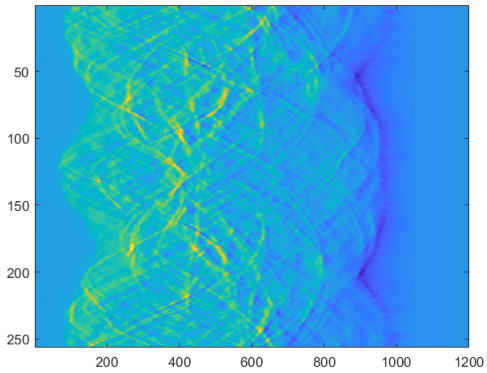

Figure 23: outcome 8

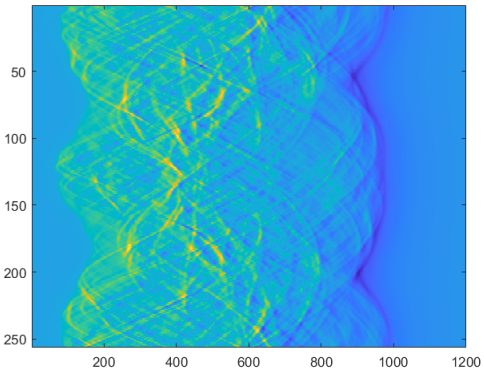

Figure 24: ground truth 8

RESULT ANALYSIS

In this part, we will estimate the feasibility of our method from these point of views. In order to show our computing result, we take the common objective evaluation index including Mean Squared Error(MSE), Peak Signal to Noise Ratio(PSNR), Mean Absolute Error(MAE) and Minkowski Distance, the definitions are as follows.

$$MSE = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}\Big[S(i,j) - S'(i,j)\Big]^2$$

$$PSNR = 10 \times \log(\frac{MAX}{MSE})$$

$$MAE = \frac{1}{MN}\sum_{i=1}^{m}\sum_{j=1}^{n}\Big|S(i,j) - S'(i,j)\Big|$$

$$D_{Minko} = \sqrt[256]{\sum_{i=1}^{M}\sum_{j=1}^{N}\Big[S(i,j) - S'(i,j)\Big]^{256}}$$

While $S(i,j)$ are from our output image and $S'(i,j)$ are from ground-truth, then following chart illustrates the index of the outcome. (Code has been attached in appendix.)

Table 1: Chart of index

Index / Outcome	$MSE/10^{-5}$	$PSNR/dB$	$MAE/10^{-3}$	$D_{Minko}/10^{-4}$
brain1	7.062	28.099	6.098	6.637
brain2	6.875	28.556	6.013	\
brain3	6.234	29.209	5.652	2.338
brain4	6.332	28.646	5.708	\
brain5	7.175	29.125	6.231	2.352
brain6	7.874	28.399	6.710	\
brain7	7.195	28.719	6.354	2.175
brain8	8.041	28.602	6.827	1.429

(The '\' means too small to read)

Our result shows that, the output image is quite similar to the ground truth, which can also be proved by visual observation. At the same time, for each outcome, our algorithm can

also achieve a relatively high PSNR, and we are convinced that they are clear enough and useable.

CONCLUSION AND DISCUSSION

In this passage, we carry out a medical image denoising algorithm based on neural network, use the mature U-net model to achieve the target of removing noise and maintain the resolution and acuteness of the image at the same time. The result shows that, for this particular task, the method performs better than many filter algorithms. It can raise PSNR, decrease MSE and remove the noise effectively, especially when the noise and has extremely similar spectrogram with the initial signal. That's the reason why it has better visual effect and is more practical and valid.

APPENDIX

[1]The code for the filter part:

```
import numpy as np
import scipy
from scipy import io
from matplotlib import pyplot as plt
from sklearn.decomposition import PCA
from scipy.fftpack import fft

X_data = [0 for i in range(8)]
X_data[0] = scipy.io.loadmat('Signal_withnoise/brain1_X.mat')['X']
X_data[1] = scipy.io.loadmat('Signal_withnoise/brain2_X.mat')['X']
X_data[2] = scipy.io.loadmat('Signal_withnoise/brain3_X.mat')['X']
X_data[3] = scipy.io.loadmat('Signal_withnoise/brain4_X.mat')['X']
X_data[4] = scipy.io.loadmat('Signal_withnoise/brain5_X.mat')['X']
X_data[5] = scipy.io.loadmat('Signal_withnoise/brain6_X.mat')['X']
X_data[6] = scipy.io.loadmat('Signal_withnoise/brain7_X.mat')['X']
X_data[7] = scipy.io.loadmat('Signal_withnoise/brain8_X.mat')['X']

Y_data = [0 for i in range(8)]
Y_data[0] = scipy.io.loadmat('Ground_truth/brain1_Y.mat')['Y']
Y_data[1] = scipy.io.loadmat('Ground_truth/brain2_Y.mat')['Y']
Y_data[2] = scipy.io.loadmat('Ground_truth/brain3_Y.mat')['Y']
Y_data[3] = scipy.io.loadmat('Ground_truth/brain4_Y.mat')['Y']
Y_data[4] = scipy.io.loadmat('Ground_truth/brain5_Y.mat')['Y']
Y_data[5] = scipy.io.loadmat('Ground_truth/brain6_Y.mat')['Y']
Y_data[6] = scipy.io.loadmat('Ground_truth/brain7_Y.mat')['Y']
Y_data[7] = scipy.io.loadmat('Ground_truth/brain8_Y.mat')['Y']

plt.pcolormesh(X_data[0])
plt.show()

plt.pcolormesh(Y_data[0])
plt.show()

X_data[0] = np.array(X_data[0])
fft1 = scipy.fftpack.fft(X_data[0])
fft1 = abs(fft1)
plt.plot(abs(fft1))
plt.show()

pca1 = PCA(n_components=1)
pca1.fit(fft1)
pca1_ = pca1.transform(fft1)
plt.plot(pca1_)
```

```
plt.show()

Y_data[0] = np.array(Y_data[0])
fft2 = scipy.fftpack.fft(Y_data[0])
fft2 = abs(fft2)
plt.plot(fft2)
plt.show()

pca2 = PCA(n_components=1)
pca2.fit(fft2)
pca2_ = pca2.transform(fft2)
plt.plot(pca2_)
plt.show()

noise1 = abs(fft1 - fft2)
plt.plot(abs(noise1))
plt.show()
pca3 = PCA(n_components=1)
pca3.fit(noise1)
pca3_ = pca2.transform(noise1)
plt.plot(pca3_)
plt.show()
```

[2]The code for data-load part

```
import numpy as np
import scipy
from scipy import io
from matplotlib import pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F

class Dataload():
  def __init__(self):
    self.X_data = None
    self.Y_data = None
  def Load(self):
    X_data = [0 for i in range(8)]
    X_data[0] = scipy.io.loadmat('Signal_withnoise/brain1_X.mat')['X']
    X_data[1] = scipy.io.loadmat('Signal_withnoise/brain2_X.mat')['X']
    X_data[2] = scipy.io.loadmat('Signal_withnoise/brain3_X.mat')['X']
    X_data[3] = scipy.io.loadmat('Signal_withnoise/brain4_X.mat')['X']
    X_data[4] = scipy.io.loadmat('Signal_withnoise/brain5_X.mat')['X']
    X_data[5] = scipy.io.loadmat('Signal_withnoise/brain6_X.mat')['X']
```

```
        X_data[6] = scipy.io.loadmat('Signal_withnoise/brain7_X.mat')['X']
        X_data[7] = scipy.io.loadmat('Signal_withnoise/brain8_X.mat')['X']

        Y_data = [0 for i in range(8)]
        Y_data[0] = scipy.io.loadmat('Ground_truth/brain1_Y.mat')['Y']
        Y_data[1] = scipy.io.loadmat('Ground_truth/brain2_Y.mat')['Y']
        Y_data[2] = scipy.io.loadmat('Ground_truth/brain3_Y.mat')['Y']
        Y_data[3] = scipy.io.loadmat('Ground_truth/brain4_Y.mat')['Y']
        Y_data[4] = scipy.io.loadmat('Ground_truth/brain5_Y.mat')['Y']
        Y_data[5] = scipy.io.loadmat('Ground_truth/brain6_Y.mat')['Y']
        Y_data[6] = scipy.io.loadmat('Ground_truth/brain7_Y.mat')['Y']
        Y_data[7] = scipy.io.loadmat('Ground_truth/brain8_Y.mat')['Y']

        self.X_data = X_data
        self.Y_data = Y_data
        self.len_X = len(X_data)

        return self.X_data, self.Y_data, self.len_X
```

[3]The code for data-set and data-import part

```
import numpy as np
from Project_Analogy.dataload import Dataload

class DataSet():
  def __init__(self):
    super().__init__()
    self.transform = True
    dataload = Dataload()
    self.X_data, self.Y_data, self.len_X = dataload.Load()

  def eval(self):
    self.transform = True

  def train(self):
    self.transform = True

  def __getitem__(self, index):
    index = index % 8
    x_brain = self.X_data[index]
    y_brain = self.Y_data[index]
    x_brain, y_brain = self.preprocess(x_brain, y_brain)
    x_brain = x_brain.reshape(1, x_brain.shape[1])
    y_brain = y_brain.reshape(1, y_brain.shape[1])
    return x_brain, y_brain
```

```
  def preprocess(self, x_data, y_data):
    if self.transform:
      i = np.random.randint(0, 256)
      brain_train = []
      truth_train = []
      brain_train.append(x_data[i])
      truth_train.append(y_data[i])
      x_data = np.array(brain_train)
      y_data = np.array(truth_train)
    return x_data, y_data

  def __len__(self):
    return self.len_X
```

[4] The code for U-net model

```
import torch
import torch.nn as nn
import torch.utils.data

class DoubleConv(nn.Module):
  def __init__(self,in_c,out_c):
    super().__init__()
    self.doubleconv=nn.Sequential(
      nn.Conv1d(in_c,out_c,kernel_size=3,padding=1),
      nn.BatchNorm1d(out_c),
      nn.ReLU(),
      nn.Conv1d(out_c,out_c,kernel_size=3,padding=1),
      nn.BatchNorm1d(out_c),
      nn.ReLU()
    )

  def forward(self,x):
    return self.doubleconv(x)

class Down(nn.Module):
  def __init__(self,in_c,out_c):
    super().__init__()
    self.down=nn.Sequential(
      nn.MaxPool1d(2),
      DoubleConv(in_c,out_c)
    )

  def forward(self,x):
```

```
        return self.down(x)

    class Up(nn.Module):
      def __init__(self,in_c,out_c):
        super().__init__()
        self.up=nn.Upsample(scale_factor=2, mode='linear', align_corners=True)
        self.conv=DoubleConv(in_c,out_c)

      def forward(self,x1,x2):
        x1=self.up(x1)
        x=torch.cat([x2, x1], dim=1)
        return self.conv(x)

    class Unet(nn.Module):
      def __init__(self,class_num,channel_num):
        super().__init__()
        self.conv1=DoubleConv(channel_num,16)
        self.down1=Down(16,32)
        self.down2=Down(32,64)
        self.down3=Down(64,128)
        self.down4=Down(128,128)
        self.up1=Up(256,64)
        self.up2=Up(128,32)
        self.up3=Up(64,16)
        self.up4=Up(32,16)
        self.out=nn.Conv1d(16,class_num,kernel_size=1)

      def forward(self,x):
        x1=self.conv1(x)
        x2=self.down1(x1)
        x3=self.down2(x2)
        x4=self.down3(x3)
        x5=self.down4(x4)
        x=self.up1(x5,x4)
        x=self.up2(x,x3)
        x=self.up3(x,x2)
        x=self.up4(x,x1)
        x=self.out(x)
        return x
```

[5]The code for the training part

```
import torch.nn as nn
import torch.utils.data
from Project_Analogy.Model import Unet
```

```
from Project_Analogy.dataset import DataSet

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = Unet(class_num=1, channel_num=1)
model.to(device=device)
model.train()

epochs = 50000
batch_size = 128
lr = 0.00001

optimizer = torch.optim.Adam(model.parameters())
loss_func = nn.MSELoss(reduction="mean")
dataset = DataSet()
dataset.train()
loader = torch.utils.data.DataLoader(dataset = dataset, batch_size=batch_size, shuffle=False)

for epoch in range(epochs):
  for x_data, y_data in loader:
    optimizer.zero_grad()
    x_data = x_data.to(device=device, dtype=torch.float32)
    y_data = y_data.to(device=device, dtype=torch.float32)
    pred = model(x_data)
    loss = loss_func(pred, y_data)
    loss.backward()
    optimizer.step()
  if epoch % 10 == 0 and epoch > 5:
    torch.save(model.state_dict(), str(epoch)+'model.pth')
  print("epoch", epoch, "loss:", loss.item())

torch.save(model.state_dict(), 'model.pth')
```

[6]The code for the test dataset

```
from Project_Analogy.dataload import Dataload

class DataSet():
  def __init__(self):
    super().__init__()
    self.transform = True
    dataload = Dataload()
    self.X_data, self.Y_data, self.len_X = dataload.Load()

  def eval(self):
    self.transform = True
```

```
  def train(self):
    self.transform = True

  def __len__(self):
    return self.len_X
```

[7]The code for the test part:

```
import torch.nn as nn
import torch.utils.data
import numpy as np
from scipy import io
import os
import glob
from Project_Analogy.Model import Unet
from Project_Analogy.dataload import Dataload

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = Unet(class_num=1, channel_num=1)
model.load_state_dict(torch.load(r"D:\19023\PycharmProjects\Machine_Learning\Project_Analogy\
                                                    model.pth"))
model.to(device=device)
model.eval()
loss_func = nn.MSELoss(reduction="mean")
dataload = Dataload()
X_data, Y_data, length = dataload.Load()
print(len(X_data))

Truth_pred = []
Truth_pred_dict = []

total_path = "D:/19023/PycharmProjects/Machine_Learning/Project_Analogy/Outcome"
outcome_path = glob.glob(os.path.join(total_path, "brain*.mat"))

loss = 0.0
i = 0
for i in range(length):
  brain_test = np.array(X_data[i])
  brain_truth = np.array(Y_data[i])

  brain_test = brain_test.reshape(256, 1, 1200)
  brain_truth = brain_truth.reshape(256, 1, 1200)

  brain_test = torch.from_numpy(brain_test)
```

```
    brain_truth = torch.from_numpy(brain_truth)

    brain_test = brain_test.to(device=device, dtype=torch.float32)
    brain_truth = brain_truth.to(device=device, dtype=torch.float32)

    brain_pred = model(brain_test)
    loss = loss_func(brain_pred, brain_truth)
    loss = loss.detach().cpu().numpy()
    brain_pred = brain_pred.detach().cpu().numpy()
    brain_pred = brain_pred.reshape(256, 1200)
    Truth_pred = brain_pred.tolist()
    print("loss:", loss.item())
    io.savemat(outcome_path[i], {'Y': Truth_pred})
```

[8]The code for the demonstration part:

```
import scipy
from scipy import io
from matplotlib import pyplot as plt

Y_data = [0 for i in range(8)]
Y_data[0] = scipy.io.loadmat('Outcome/brain1_Y.mat')['Y']
Y_data[1] = scipy.io.loadmat('Outcome/brain2_Y.mat')['Y']
Y_data[2] = scipy.io.loadmat('Outcome/brain3_Y.mat')['Y']
Y_data[3] = scipy.io.loadmat('Outcome/brain4_Y.mat')['Y']
Y_data[4] = scipy.io.loadmat('Outcome/brain5_Y.mat')['Y']
Y_data[5] = scipy.io.loadmat('Outcome/brain6_Y.mat')['Y']
Y_data[6] = scipy.io.loadmat('Outcome/brain7_Y.mat')['Y']
Y_data[7] = scipy.io.loadmat('Outcome/brain8_Y.mat')['Y']

plt.pcolormesh(Y_data[0])
plt.show()
```

[9] The code for the result estimating part

```
import numpy as np
import scipy
from scipy import io
from math import sqrt
import math
from matplotlib import pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```
# import the ground truth
Y_data = [0 for i in range(8)]
Y_data[0] = scipy.io.loadmat('Ground_truth/brain1_Y.mat')['Y']
Y_data[1] = scipy.io.loadmat('Ground_truth/brain2_Y.mat')['Y']
Y_data[2] = scipy.io.loadmat('Ground_truth/brain3_Y.mat')['Y']
Y_data[3] = scipy.io.loadmat('Ground_truth/brain4_Y.mat')['Y']
Y_data[4] = scipy.io.loadmat('Ground_truth/brain5_Y.mat')['Y']
Y_data[5] = scipy.io.loadmat('Ground_truth/brain6_Y.mat')['Y']
Y_data[6] = scipy.io.loadmat('Ground_truth/brain7_Y.mat')['Y']
Y_data[7] = scipy.io.loadmat('Ground_truth/brain8_Y.mat')['Y']

# import the prediction
Y_pred = [0 for j in range(8)]
Y_pred[0] = scipy.io.loadmat('Outcome/brain1_Y.mat')['Y']
Y_pred[1] = scipy.io.loadmat('Outcome/brain2_Y.mat')['Y']
Y_pred[2] = scipy.io.loadmat('Outcome/brain3_Y.mat')['Y']
Y_pred[3] = scipy.io.loadmat('Outcome/brain4_Y.mat')['Y']
Y_pred[4] = scipy.io.loadmat('Outcome/brain5_Y.mat')['Y']
Y_pred[5] = scipy.io.loadmat('Outcome/brain6_Y.mat')['Y']
Y_pred[6] = scipy.io.loadmat('Outcome/brain7_Y.mat')['Y']
Y_pred[7] = scipy.io.loadmat('Outcome/brain8_Y.mat')['Y']

# Calculate
def MSE(y_pred, y_true):
  return np.sum((y_true - y_pred)**2)/len(y_true)
def RMSE(y_pred, y_true):
  return sqrt(np.sum((y_true - y_pred)**2)/len(y_true))
def MAE(y_pred, y_true):
  return np.sum(abs(y_true - y_pred))/len(y_true)
def R2(y_pred, y_true):
  return 1 - MSE(y_true, y_pred)/np.var(y_true)
def PSNR(y_pred, y_true):
  return 10 * math.log10(((max(y_pred))**2) / MSE(y_true, y_pred))
def Minkowski_Dis(y_pred, y_true):
  return (np.sum((y_true - y_pred)**256))**(1/256)

MSE_list = []
RMSE_list = []
MAE_list = []
R2_list = []
PSNR_list = []
Minkowski_Dis_list = []

for j in range(0, 8):
  for i in range(0, 256):
```

```
        MSE_list.append(MSE(Y_pred[j][i], Y_data[j][i]))
        RMSE_list.append(RMSE(Y_pred[j][i], Y_data[j][i]))
        MAE_list.append(MAE(Y_pred[j][i], Y_data[j][i]))
        R2_list.append(R2(Y_pred[j][i], Y_data[j][i]))
        PSNR_list.append(PSNR(Y_pred[j][i], Y_data[j][i]))
        Minkowski_Dis_list.append(Minkowski_Dis(Y_pred[j][i], Y_data[j][i]))
    print("------------------", "brain", j,"--------------------")
    print("MSE:", "brain", j, np.sum(MSE_list)/len(MSE_list))
    print("RMSE:", "brain", j, np.sum(RMSE_list)/len(RMSE_list))
    print("MAE:", "brain", j, np.sum(MAE_list)/len(MAE_list))
    print("R2", "brain", j, np.sum(R2_list)/len(R2_list))
    print("PSNR", "brain", j, np.sum(PSNR_list)/len(PSNR_list))
    print("Minkowski_Dis", "brain", j, np.sum(Minkowski_Dis_list)/len(Minkowski_Dis_list))
    del MSE_list[:]
    del RMSE_list[:]
    del MAE_list[:]
    del R2_list[:]
    del PSNR_list[:]
    del Minkowski_Dis_list[:]
```