Salt Identification and Segmentation

Lu Bai bai.lu2@husky.neu.edu
Shuwan Zhang zhang.shuwa@husky.neu.edu

Abstract

The identification of salt bodies plays an important role in geological research and exploration of oil and gas, but now professionals still need to manually distinguish seismic images. This project aims to build an effective model through deep learning to quickly identify areas of the salt bodies in seismic images.

Our dataset is a set of images chosen at various locations chosen at random by using reflection seismology in the subsurface. After data exploratory, we implemented U-Net and CNN models using Keras to process and capture features of our train data. After comparing the prediction results of the two models, we select one of the excellent ones to process the test dataset and obtain the salt bodies with higher accuracy. In this paper, we will elaborate how seismic images can be used in recognizing salt bodies step by step.

The results obtained in this research is showing the best accuracy of our U-Net model is about 0.9221 and the best accuracy of our CNN model is about 0.8507. So, we choose the U-Net model to predict our dataset data.

Keywords: Salt bodies, U-NET, CNN, Deep learning

Introduction

There are also huge salt deposits on the earth beneath the surface of the earth where oil and gas are concentrated. In the early stages of oil and gas exploration, due to the drilling risks associated with salty land, people avoid salty land at all costs. But unfortunately, it is still difficult to understand the exact location of large salt deposits. Professional seismic imaging still requires experts to interpret salt, so this leads to the need for a lot of manpower and the subjectivity of manual identification. To help the development of geological research and exploration of oil and gas, we start to collect the seismic data and do some research. In our project, we use two deep learning methods to identify salt bodies deposits beneath the Earth's surface. The dataset provided with some seismic images that are produced from imaging the reflection coming from rock boundaries.

Background

Typically, seismic data is collected using reflection seismology. This method requires sensors to record the reflections of the underground rock interface and then create a three-dimensional view of the interior of the Earth. Reflection seismology [1] is similar to X-ray, sonar and echolocation. From the seismic data, a seismic image of the reflection from the rock boundary can be obtained. Earthquake images show the boundaries between different rock types. Although seismic images show rock boundaries, they do not say too much about the rock itself. Some rocks are very easy to distinguish, while others are difficult.

There are many areas on the planet that contain salt, and one of the biggest challenges of seismic imaging is the identification of salt components in the underground. Salt is actually very simple but has unrecognizable features. Usually, the salt density is 2.14 g/cc [2], which is lower than most surrounding rocks. The salt has an earthquake speed of 4.5 km/s [3] and is also faster than the surrounding rocks. This difference produces a significant reflection at the salt-sediment interface. Salt is amorphous rock and does not have much internal structure. This means that there is not much reflectivity n unless there is sediment in the salt. Therefore, an unusually high seismic velocity of the salt may cause problems with seismic imaging.

Dataset

The dataset contains 22000 images of seismic in total from different depth all over the world. Among them, 4000 images are train dataset with mask images and 18000 are test set. The dataset also contains depth csv of each images and the RLE data csv of mask images.

The dataset has the following distribution:

1. There are 22000 images in total with id and each image has a depth data in depth.csv
2. Train set contains roughly 20% of the total images and each image has one mask image to fit
3. Mask images has rle.csv to record data

Sample images from the dataset are shown below:

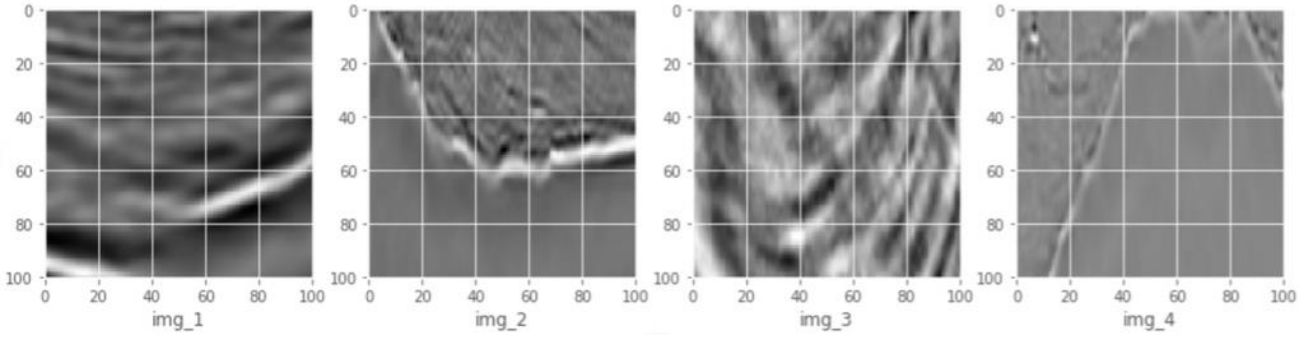

Figure 2.1 Seismic Images

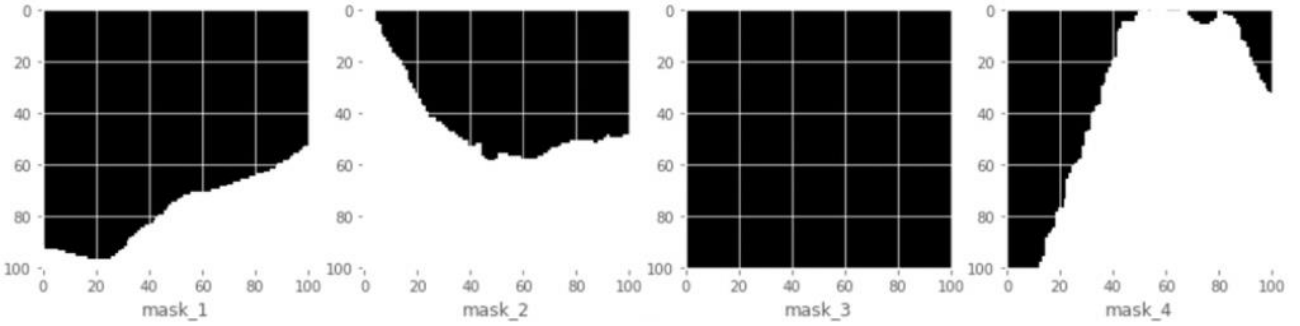

Figure 2.2 Mask Images

id	rle_mask
575d24d81d	
a266a2a9df	5051 5151
75efad62c1	9 93 109 94 210 94 310 95 411 95 511 96 612 96 712 97 812 98 913 98 1015 97 1116 97 1216 98 1316 99 1416 8786
34e51dba6a	48 54 149 54 251 53 353 52 455 51 557 50 659 49 762 47 864 46 966 45 1068 44 1171 42 1273 41 1376 39 1478 38 1581 36 1683 35
4875705fb0	1111 1 1212 1 1313 1 1414 1 1514 2 1615 2 1716 2 1817 2 1918 2 2018 3 2119 3 2220 3 2321 3 2422 3 2523 3 2624 3 2725 3 2826 3
782ae9b7e7	1 1815 1819 90 1920 81 2021 73 2122 64 2223 55 2324 46 2425 36 2526 25 2627 13 2728 1
9842f69f8d	
aa94cfb806	1 28 102 28 203 29 304 30 405 32 506 33 607 34 708 35 809 35 910 36 1011 37 1112 37 1213 38 1314 39 1415 40 1516 41 1617 42 1
50d3073821	1 2121 9293 909
28f865caaa	
b5e1371b3b	75 27 175 28 275 29 374 31 474 32 574 33 674 34 773 36 874 36 974 37 1074 38 1174 39 1274 40 1375 40 1475 41 1575 42 1676 42

Figure 2.3 RLE Data

id	z
4ac19fb269	306
1825fadf99	157
f59821d067	305
5b435fad9d	503
e340e7bfca	783
2ffea0c397	429
6cf284fb9e	600
d0244d6c38	51

Figure2.4 Depth Data

Model Selection

3-1. Convolutional Neural Networks (CNN / ConvNets)

Convolutional Neural Networks are a category of Neural Networks that have proven very effective in areas such as image and video recognition, recommender systems, image classification, medical image analysis, and natural language processing [4].

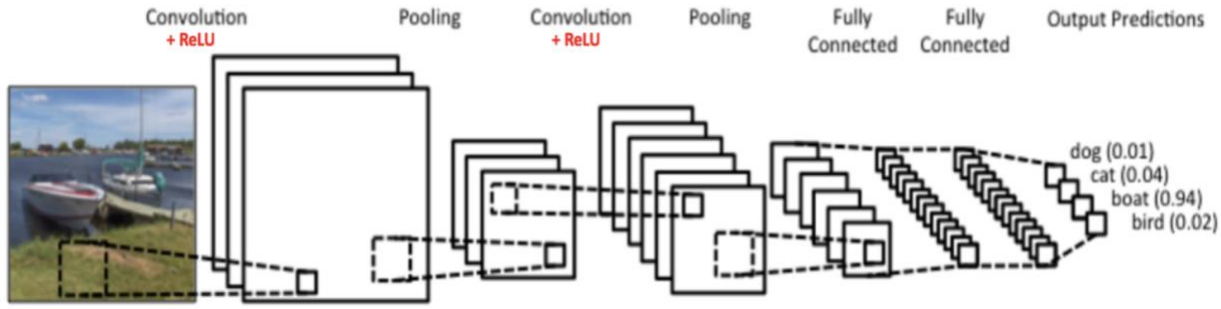

Figure 3-1

Figure 3-1shows that the structure of CNN architecture. We can see that there are three main types of layers to build CNN architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer. These layers transform the input volume into an output volume through a differentiable function [4].

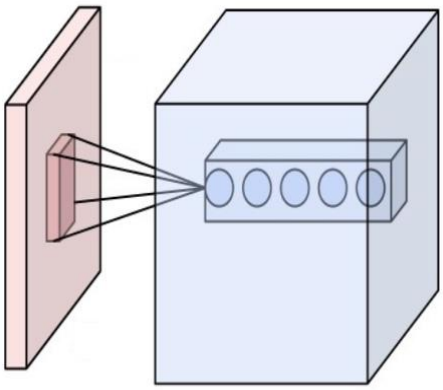

Figure 3-2

Figure 3-2 shows the structure of convolutional layer. Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The parameters of a layer consist of a set of learnable filters (or kernels) that can be extended to the entire depth of the input volume. During forward transfer, each filter convolves over the width and height of the input volume, calculates the dot product between the filter's entries and the input, and produces a two-dimensional activation map

of the filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input. [1]

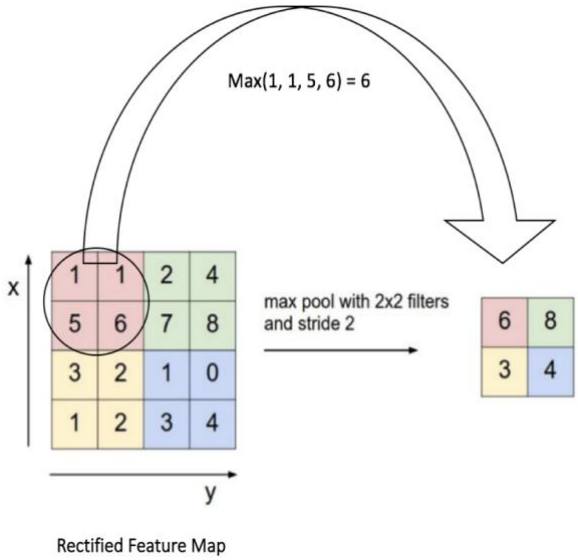

Figure 3-3

Figure 3-3 shows that the working principle of max pooling. The function of Pooling is to progressively reduce the spatial size of the input representation. In particular, pooling can reduce the number of parameters and computations in the network in order to control overfitting. It also makes the input representations smaller and more manageable. Max pooling and Average pooling are commonly used. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Average pooling, which uses the average value from each of a cluster of neurons at the prior layer [8].

The fully connected layer follows several convolutional and max pooling layers. Neurons in a fully connected layer have connections to all activations in the previous layer. Their activations can thus be computed as an affine transformation, with matrix multiplication followed by a bias offset.

3.2. U-Net: Convolutional Networks for Biomedical Image Segmentation

U-Net is a Fully Convolutional Network (FCN) that does fast and precise image segmentation. Its goal is then to predict each pixel's class. Compared to the CNN, the U-Net combines the location information from the downsampling path with the contextual information in upsampling path to finally obtain a general information combining localization and context, which is necessary to predict a good segmentation map. There is also not dense layer in U-Net model, so images of different sizes can be used as input.

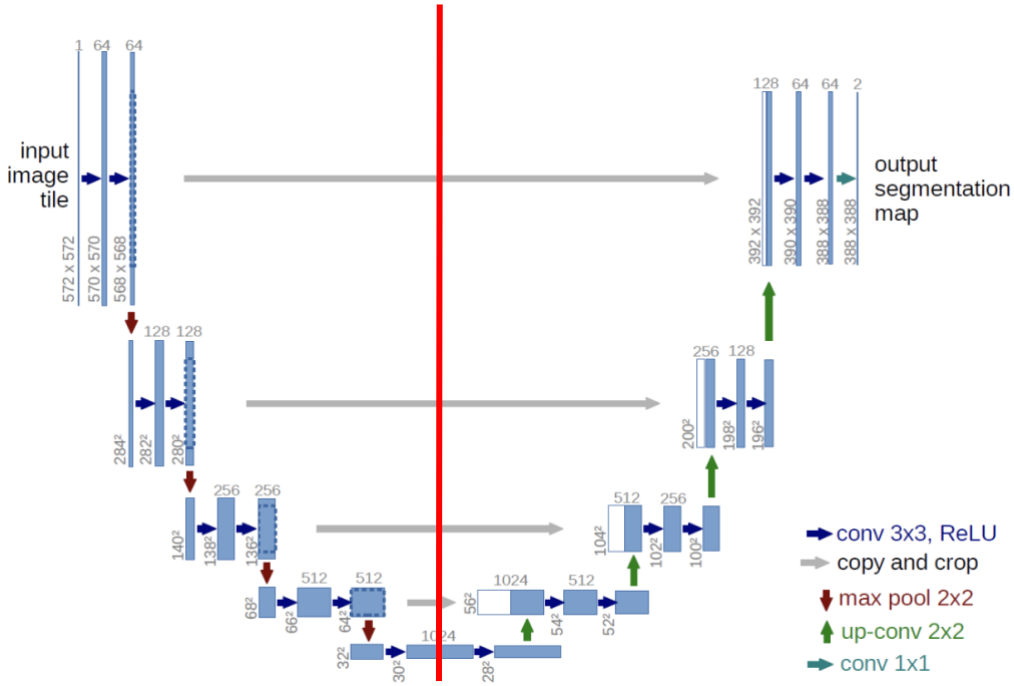

Figure 3-4

Figure 3-4 shows the architecture of U-Net. We can see that the architecture is separated in 2 parts: a contracting path (left side) and an expansive path (right side). The contracting path follows the typical architecture of a convolutional network. It is composed of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for down sampling. At each downsampling step, we double the number of feature channels. Every step in the expansive path is composed of an upsampling of the feature map followed by a 2x2 convolution (‘up-convolution’) that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. At the final layer a 1x1 convolution is used to map each 64 components feature vector to the desired number of classes.

Methodology

For this project, we have designed our own method using both U-net and CNN. The following steps discuss about how our dataset processed and model trained.

4.1 Process input data

The images of the dataset are placed in the folder containing its own id. Since images are in gray color, we set the image channel as one. To have a quality data for training models, we resize all images to same size.

Plus, join depth csv with RLE csv by CONCAT function to prepare data for analysis of depth and salt.

4.2 Calculate salt ratio

Our dataset has RLE data which is a simple yet efficient format for storing binary masks which only has 2 colors. And each RLE data links to one mask image. Since RLE data can be converted to the image binary data, we write a method to calculate salt ratio. Specifically, after the RLE data is converted into image data, an array of 101 times 101 is formed. When the color of a pixel of a picture is black, the number in array is 0, which means that it is not the position of the salt zone. Otherwise, it has 255 to represent white which is the salt body. A salt ratio of one image is the number of 255 by the UNIQUE method and divide it with the total pixel of image.

4.3 The relationship between depth and salt ratio

To check if it has some relationship between depth and salt ratio, we use linear model to train them. Figure 4-1 and Figure4-2 show the result of this liner model.

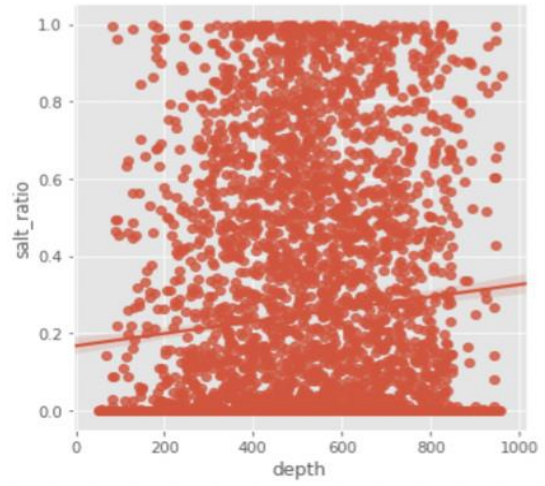

Figure 4-1

OLS Regression Results

Dep. Variable:	salt_ratio	**R-squared:**	0.359
Model:	OLS	**Adj. R-squared:**	0.359
Method:	Least Squares	**F-statistic:**	2244.
Date:	Wed, 12 Dec 2018	**Prob (F-statistic):**	0.00
Time:	17:01:45	**Log-Likelihood:**	-1154.6
No. Observations:	4000	**AIC:**	2311.
Df Residuals:	3999	**BIC:**	2317.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
depth	0.0004	9.32e-06	47.373	0.000	0.000	0.000

Omnibus:	416.062	**Durbin-Watson:**	1.990
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	536.831
Skew:	0.884	**Prob(JB):**	2.68e-117
Kurtosis:	2.686	**Cond. No.**	1.00

Figure 4-2

The left figure shows the relation of depth and salt_ratio and the right figure shows the summary of this model. As results shown above, the coefficient of linear model is 0.00015861 and the intercept is 0.16759867. Plus, R-squared is only 0.359. We would like to say there only have little relationship with depth and salt_ratio. So next steps we will ignore the depth of each image to train our model.

4.4 Build U-Net model and CNN model

4.4.1 U-Net Model Summary

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 128, 128, 1)	0	
conv2d_1 (Conv2D)	(None, 128, 128, 16)	160	input_layer[0][0]
batch_normalization_1 (BatchNor	(None, 128, 128, 16)	64	conv2d_1[0][0]
activation_1 (Activation)	(None, 128, 128, 16)	0	batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 16)	2320	activation_1[0][0]
batch_normalization_2 (BatchNor	(None, 128, 128, 16)	64	conv2d_2[0][0]
activation_2 (Activation)	(None, 128, 128, 16)	0	batch_normalization_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 16)	0	activation_2[0][0]
dropout_1 (Dropout)	(None, 64, 64, 16)	0	max_pooling2d_1[0][0]
conv2d_3 (Conv2D)	(None, 64, 64, 32)	4640	dropout_1[0][0]
batch_normalization_3 (BatchNor	(None, 64, 64, 32)	128	conv2d_3[0][0]
activation_3 (Activation)	(None, 64, 64, 32)	0	batch_normalization_3[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 32)	9248	activation_3[0][0]
batch_normalization_4 (BatchNor	(None, 64, 64, 32)	128	conv2d_4[0][0]
activation_4 (Activation)	(None, 64, 64, 32)	0	batch_normalization_4[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 32)	0	activation_4[0][0]
dropout_2 (Dropout)	(None, 32, 32, 32)	0	max_pooling2d_2[0][0]
conv2d_5 (Conv2D)	(None, 32, 32, 64)	18496	dropout_2[0][0]
batch_normalization_5 (BatchNor	(None, 32, 32, 64)	256	conv2d_5[0][0]
activation_5 (Activation)	(None, 32, 32, 64)	0	batch_normalization_5[0][0]
conv2d_6 (Conv2D)	(None, 32, 32, 64)	36928	activation_5[0][0]
batch_normalization_6 (BatchNor	(None, 32, 32, 64)	256	conv2d_6[0][0]
activation_6 (Activation)	(None, 32, 32, 64)	0	batch_normalization_6[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 64)	0	activation_6[0][0]
dropout_3 (Dropout)	(None, 16, 16, 64)	0	max_pooling2d_3[0][0]
conv2d_7 (Conv2D)	(None, 16, 16, 128)	73856	dropout_3[0][0]
batch_normalization_7 (BatchNor	(None, 16, 16, 128)	512	conv2d_7[0][0]
activation_7 (Activation)	(None, 16, 16, 128)	0	batch_normalization_7[0][0]
conv2d_8 (Conv2D)	(None, 16, 16, 128)	147584	activation_7[0][0]
batch_normalization_8 (BatchNor	(None, 16, 16, 128)	512	conv2d_8[0][0]
activation_8 (Activation)	(None, 16, 16, 128)	0	batch_normalization_8[0][0]

max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 128)	0	activation_8[0][0]
dropout_4 (Dropout)	(None, 8, 8, 128)	0	max_pooling2d_4[0][0]
conv2d_9 (Conv2D)	(None, 8, 8, 256)	295168	dropout_4[0][0]
batch_normalization_9 (BatchNor	(None, 8, 8, 256)	1024	conv2d_9[0][0]
activation_9 (Activation)	(None, 8, 8, 256)	0	batch_normalization_9[0][0]
conv2d_10 (Conv2D)	(None, 8, 8, 256)	590080	activation_9[0][0]
batch_normalization_10 (BatchNo	(None, 8, 8, 256)	1024	conv2d_10[0][0]
activation_10 (Activation)	(None, 8, 8, 256)	0	batch_normalization_10[0][0]
conv2d_transpose_1 (Conv2DTrans	(None, 16, 16, 128)	295040	activation_10[0][0]
concatenate_1 (Concatenate)	(None, 16, 16, 256)	0	conv2d_transpose_1[0][0] activation_8[0][0]
dropout_5 (Dropout)	(None, 16, 16, 256)	0	concatenate_1[0][0]
conv2d_11 (Conv2D)	(None, 16, 16, 128)	295040	dropout_5[0][0]
batch_normalization_11 (BatchNo	(None, 16, 16, 128)	512	conv2d_11[0][0]
activation_11 (Activation)	(None, 16, 16, 128)	0	batch_normalization_11[0][0]
conv2d_12 (Conv2D)	(None, 16, 16, 128)	147584	activation_11[0][0]
batch_normalization_12 (BatchNo	(None, 16, 16, 128)	512	conv2d_12[0][0]
activation_12 (Activation)	(None, 16, 16, 128)	0	batch_normalization_12[0][0]
conv2d_transpose_2 (Conv2DTrans	(None, 32, 32, 64)	73792	activation_12[0][0]
concatenate_2 (Concatenate)	(None, 32, 32, 128)	0	conv2d_transpose_2[0][0] activation_6[0][0]
dropout_6 (Dropout)	(None, 32, 32, 128)	0	concatenate_2[0][0]
conv2d_13 (Conv2D)	(None, 32, 32, 64)	73792	dropout_6[0][0]
batch_normalization_13 (BatchNo	(None, 32, 32, 64)	256	conv2d_13[0][0]
activation_13 (Activation)	(None, 32, 32, 64)	0	batch_normalization_13[0][0]
conv2d_14 (Conv2D)	(None, 32, 32, 64)	36928	activation_13[0][0]
batch_normalization_14 (BatchNo	(None, 32, 32, 64)	256	conv2d_14[0][0]
activation_14 (Activation)	(None, 32, 32, 64)	0	batch_normalization_14[0][0]
conv2d_transpose_3 (Conv2DTrans	(None, 64, 64, 32)	18464	activation_14[0][0]
concatenate_3 (Concatenate)	(None, 64, 64, 64)	0	conv2d_transpose_3[0][0] activation_4[0][0]
dropout_7 (Dropout)	(None, 64, 64, 64)	0	concatenate_3[0][0]
conv2d_15 (Conv2D)	(None, 64, 64, 32)	18464	dropout_7[0][0]
batch_normalization_15 (BatchNo	(None, 64, 64, 32)	128	conv2d_15[0][0]

activation_15 (Activation)	(None, 64, 64, 32)	0	batch_normalization_15[0][0]
conv2d_16 (Conv2D)	(None, 64, 64, 32)	9248	activation_15[0][0]
batch_normalization_16 (BatchNo	(None, 64, 64, 32)	128	conv2d_16[0][0]
activation_16 (Activation)	(None, 64, 64, 32)	0	batch_normalization_16[0][0]
conv2d_transpose_4 (Conv2DTrans	(None, 128, 128, 16)	4624	activation_16[0][0]
concatenate_4 (Concatenate)	(None, 128, 128, 32)	0	conv2d_transpose_4[0][0] activation_2[0][0]
dropout_8 (Dropout)	(None, 128, 128, 32)	0	concatenate_4[0][0]
conv2d_17 (Conv2D)	(None, 128, 128, 16)	4624	dropout_8[0][0]
batch_normalization_17 (BatchNo	(None, 128, 128, 16)	64	conv2d_17[0][0]
activation_17 (Activation)	(None, 128, 128, 16)	0	batch_normalization_17[0][0]
conv2d_18 (Conv2D)	(None, 128, 128, 16)	2320	activation_17[0][0]
batch_normalization_18 (BatchNo	(None, 128, 128, 16)	64	conv2d_18[0][0]
activation_18 (Activation)	(None, 128, 128, 16)	0	batch_normalization_18[0][0]
conv2d_19 (Conv2D)	(None, 128, 128, 1)	17	activation_18[0][0]

Total params: 2,164,305
Trainable params: 2,161,361
Non-trainable params: 2,944

The network architecture is illustrated in the summary of this U-Net model. It is composed of a contracting path and an expansive path. The contracting path aims to capture context and a symmetric expansive path which helps to precise localization.

In the contracting path, we use convolutional layers to capture features, a normalization function to make network train faster, the pooling layer to further capture the remarkable features and the output layer to reduce overfitting. Furthermore, in expansive path, we use counter-convolutional layer and concatenation to combine all feature maps into one prediction mask based on the nucleus position.

4.4.2 CNN Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
NormalizeInput (BatchNormali (None, None, None, 1)     4
_________________________________________________________________
conv2d_20 (Conv2D)           (None, None, None, 8)     80
_________________________________________________________________
conv2d_21 (Conv2D)           (None, None, None, 16)    1168
_________________________________________________________________
conv2d_22 (Conv2D)           (None, None, None, 32)    4640
_________________________________________________________________
conv2d_23 (Conv2D)           (None, None, None, 64)    18496
_________________________________________________________________
dense_1 (Dense)              (None, None, None, 1)     65
=================================================================
Total params: 24,453
Trainable params: 24,451
Non-trainable params: 2
_________________________________________________________________
```

The network architecture is illustrated in the summary of this CNN model. The input data has been normalized by 'BatchNormalization' function. The simple CNN structure is composed of one input layer, four convolutional layers and one fully connected layer.

4.5 Training models

4.5.1U-Net training

The U-Net model is trained for 100 epochs. After 19 epochs, the val_loss has never improved. So, the training process has early stopped at the 20th epoch. We can calculate the best accuracy is about 0.9221.

```
Epoch 00015: val_loss did not improve from 0.24305
Epoch 16/100
3400/3400 [==============================] - 638s 188ms/step - loss: 0.1867 - acc: 0.9201 - val_loss: 0.2558 - val_ac
c: 0.8748

Epoch 00016: ReduceLROnPlateau reducing learning rate to 1e-05.

Epoch 00016: val_loss did not improve from 0.24305
Epoch 17/100
3400/3400 [==============================] - 639s 188ms/step - loss: 0.1867 - acc: 0.9195 - val_loss: 0.2560 - val_ac
c: 0.8748

Epoch 00017: val_loss did not improve from 0.24305
Epoch 18/100
3400/3400 [==============================] - 624s 184ms/step - loss: 0.1815 - acc: 0.9221 - val_loss: 0.2515 - val_ac
c: 0.8774

Epoch 00018: val_loss did not improve from 0.24305
Epoch 19/100
3400/3400 [==============================] - 603s 177ms/step - loss: 0.1870 - acc: 0.9186 - val_loss: 0.2536 - val_ac
c: 0.8759

Epoch 00019: val_loss did not improve from 0.24305
Epoch 20/100
3400/3400 [==============================] - 636s 187ms/step - loss: 0.1859 - acc: 0.9205 - val_loss: 0.2579 - val_ac
c: 0.8737

Epoch 00020: val_loss did not improve from 0.24305
Epoch 00020: early stopping
```

4.5.2 CNN training

The CNN model is trained for 100 epochs. Althouth the val_loss has been growing, the growth rate of val_loss is very slow, and accuracy has not improved much. The best accuracy of CNN model is 0.8507.

```
Epoch 00094: val_loss improved from 0.32669 to 0.32656, saving model to cnn-tgs-salt.h5
Epoch 95/100
3400/3400 [==============================] - 208s 61ms/step - loss: 0.3550 - acc: 0.8498 - val_loss: 0.3267 - val_ac
c: 0.8699

Epoch 00095: val_loss did not improve from 0.32656
Epoch 96/100
3400/3400 [==============================] - 215s 63ms/step - loss: 0.3554 - acc: 0.8499 - val_loss: 0.3266 - val_ac
c: 0.8699

Epoch 00096: val_loss improved from 0.32656 to 0.32655, saving model to cnn-tgs-salt.h5
Epoch 97/100
3400/3400 [==============================] - 223s 65ms/step - loss: 0.3557 - acc: 0.8502 - val_loss: 0.3265 - val_ac
c: 0.8699

Epoch 00097: val_loss improved from 0.32655 to 0.32653, saving model to cnn-tgs-salt.h5
Epoch 98/100
3400/3400 [==============================] - 225s 66ms/step - loss: 0.3571 - acc: 0.8493 - val_loss: 0.3266 - val_ac
c: 0.8699

Epoch 00098: val_loss did not improve from 0.32653
Epoch 99/100
3400/3400 [==============================] - 227s 67ms/step - loss: 0.3549 - acc: 0.8505 - val_loss: 0.3265 - val_ac
c: 0.8699

Epoch 00099: val_loss improved from 0.32653 to 0.32650, saving model to cnn-tgs-salt.h5
Epoch 100/100
3400/3400 [==============================] - 220s 65ms/step - loss: 0.3558 - acc: 0.8505 - val_loss: 0.3266 - val_ac
c: 0.8700

Epoch 00100: val_loss did not improve from 0.32650
```

4.6 Evaluation of models

4.6.1 U-Net evaluate

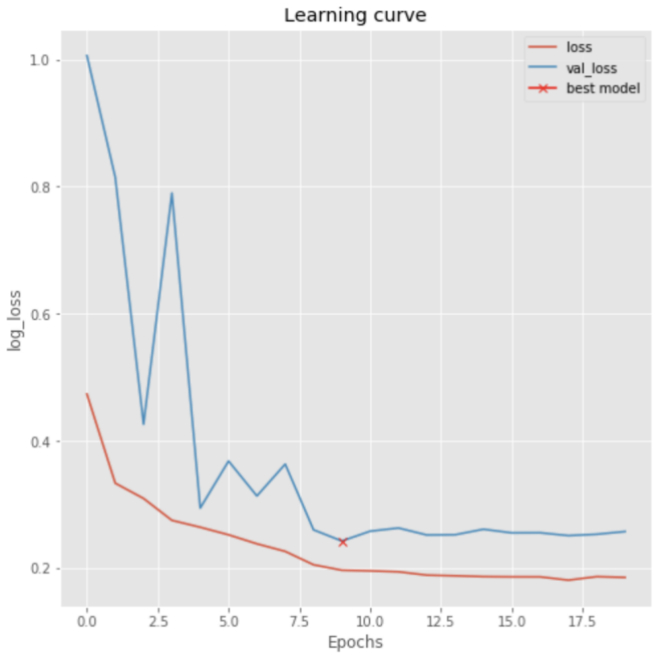

Figure 4-3

Figure 4-3 show that the ‘loss’ and ‘val_loss’ plots.

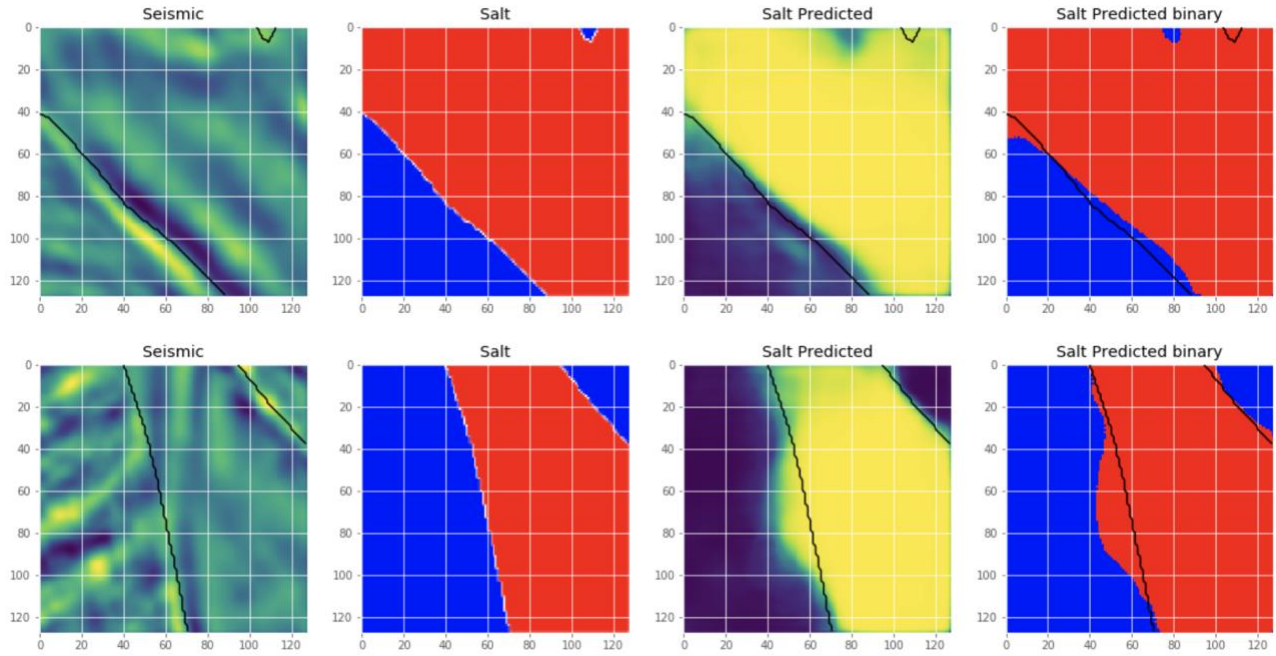

Figure 4-4

Figure 4-4 show that the ordinary image, the corresponding mask, the predicted image and the mask of predicted image using U-Net model.

4.6.1 CNN evaluate

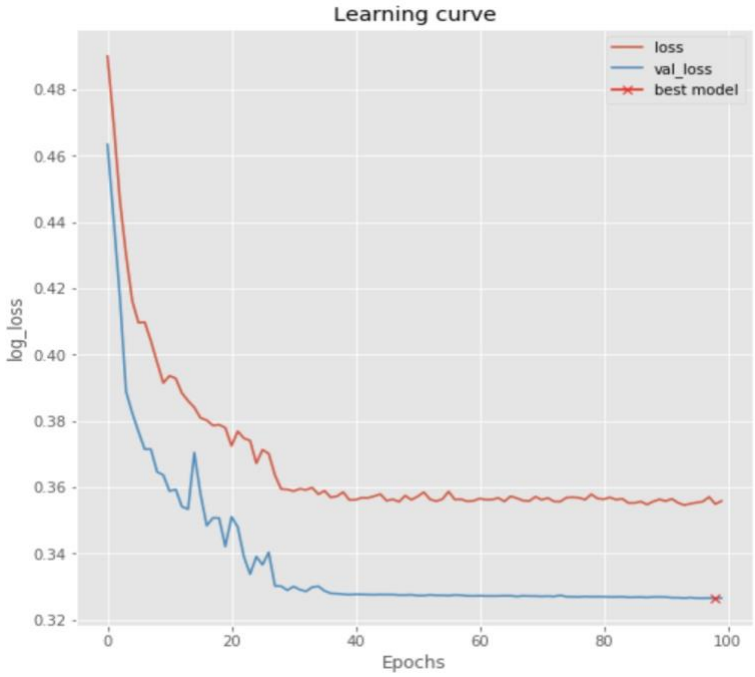

Figure 4-5

Figure 4-5 show that the ‘loss’ and ‘val_loss’ plots.

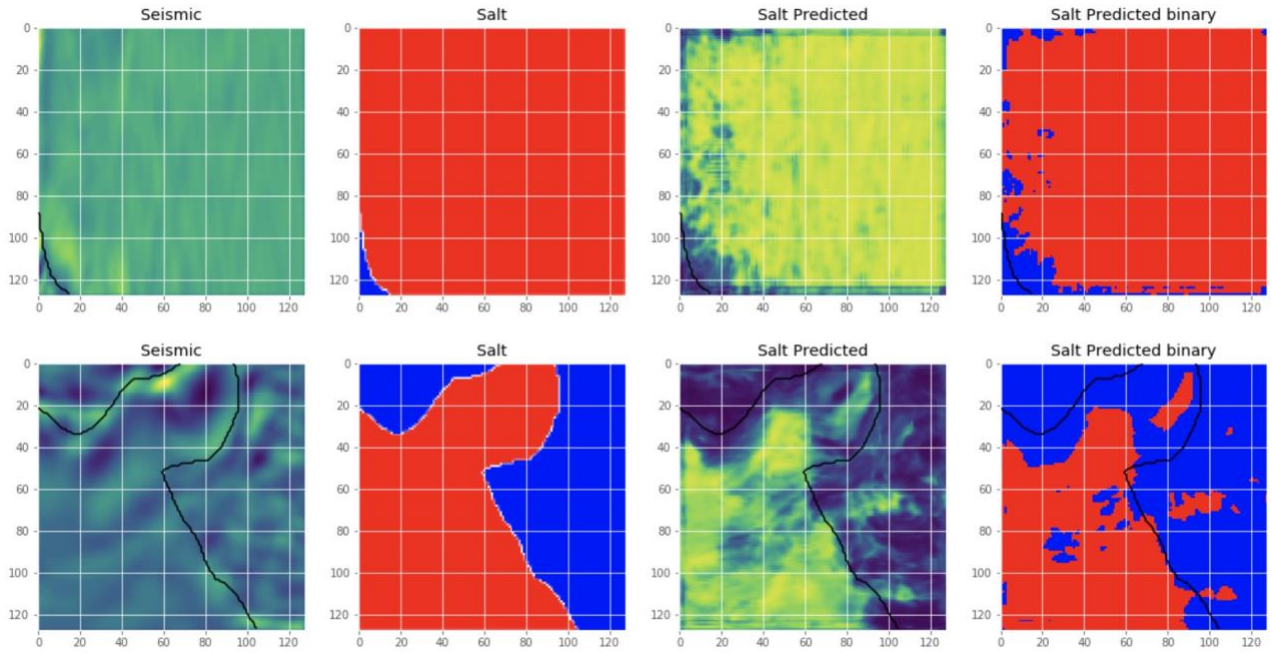

Figure 4-6

Figure 4-6 show that the ordinary image, the corresponding mask, the predicted image and the mask of predicted image using CNN model.

We use the ‘loss’, ‘accuracy’, ‘val_loss’ and ‘val_accuracy’ to evaluate these two models.

‘val_loss’ is the value of cost function for our cross validation data and ‘loss’ is the value of cost function for our training data.

Furthermore, the plots drawn above also can be the evidence to evaluate these models.

Results

After several times' experiments, we can find that the U-Net model is better than the CNN model. According to the results of these models, the accuracy of CNN model is about 0.85 and the loss of CNN model is about 0.36. These two values have shown that the CNN model has not been trained very well. The sample plots of predictions also demonstrate that the model cannot distinguish the boundary between salt area and other matter.

However, the accuracy of U-Net model is about 0.92 and the loss of U-net model is about 0.18. These two reference values indicate that the model can be used to accurately identify the salt area. We have evaluated the U-Net model with validation data. Because the accuracy of this model cannot reach 100%, there will be some errors in the process of judgment. However, some plots show that the accuracy of the judgment is very high.

In conclusion, we finally decide to use U-Net model for identify the salt area using test images.

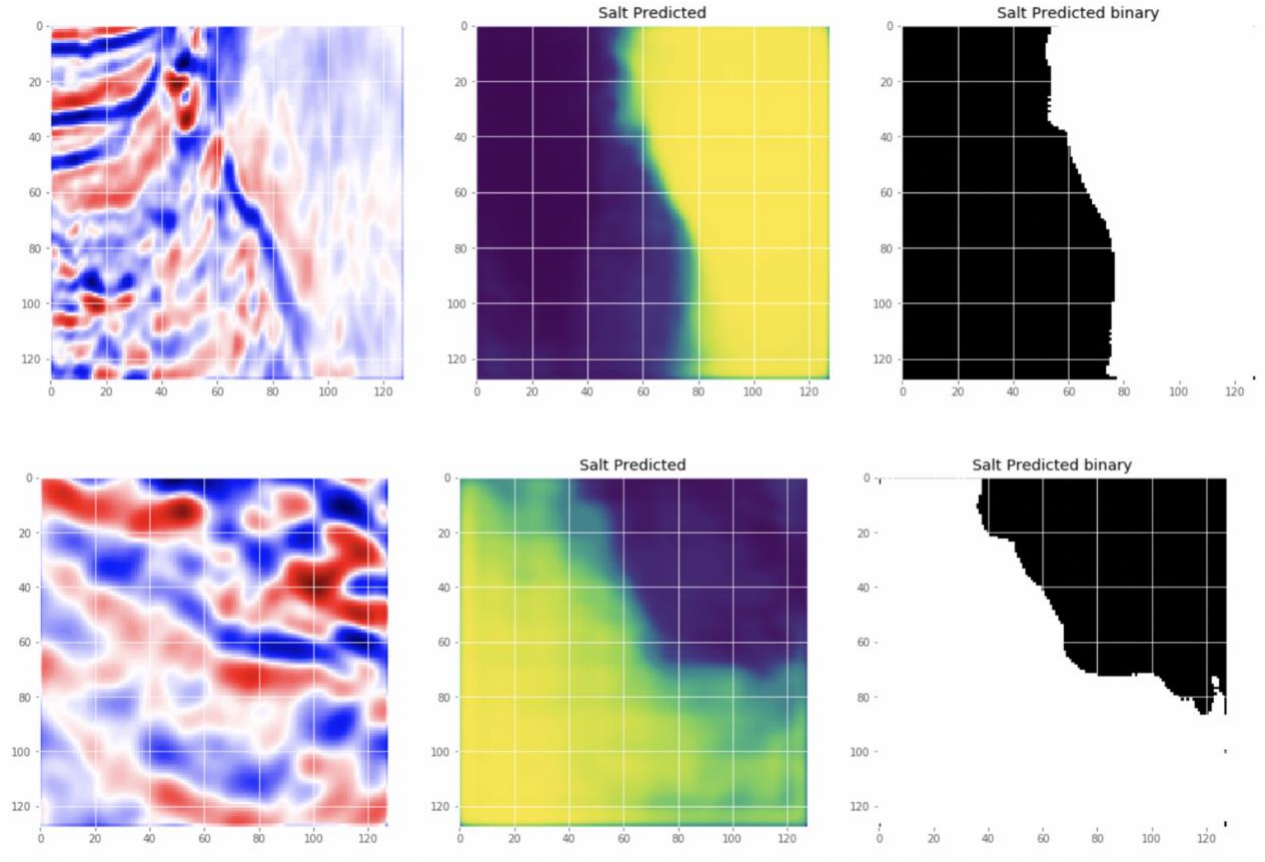

Figure 5-1

Figure 5-1 shows that the ordinary test image, the predicted image of salt area and the mask of predicted image using U-Net model.

Discussion

6.1 Our project

In our project, we both discussed a lot to make our models more efficient and more quickly. Here are some approaches we have done and changed.

Firstly, we have defined the callbacks function including EarlyStopping, ReduceLROnPlateau and ModelCheckpoint to help us evaluate our models. Early stopping is basically stopping the training once the loss starts to increase. The parameter patience represents the number of epochs before stopping once the validation loss starts to increase. Reduce learning rate when a metric has stopped improving. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of

epochs, the learning rate is reduced. The model of checkpoints will be saved with the epoch number and the validation loss in the filename after every epoch.

Secondly, we compared the difference between exist of the batch normalization or not when building the U-net model. We find with the batch normalization, our model trains much faster and more efficient. And it allows us to use much higher learning rates, which further increases the speed at which networks train.

Thirdly, when we build our CNN model, we only get the accuracy at 0.4 of data. After changing the architecture of model, we finally increase our model accuracy. In rebuilding process, one part is we changed sigmoid activation to rlu. And after test so many times with a long waiting, we draw the conclusion that rlu activation is much better than sigmoid.

Finally, one method to reduce overfitting is dropout. At each training stage, individual nodes are either "dropped out" of the net with probability or kept with probability, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. Only the reduced network is trained on the data in that stage. The removed nodes are then reinserted into the network with their original weights.

6.2 Others in kernel

Since our dataset from Kaggle, we saw others' effort. For this dataset, some of them get a high accuracy in u-net model, but they seldom predict the dataset that give me a question about if their model has an overfitting issue. Plus, we find some of them using depth as a parameter to train their model, but we found out their only has a little relationship with depth and salt bodies which may doing the useless work for this dataset. For us, we choose to build our own models. From our point of view, only one model is not so efficient to evidence it's the best one. After learning neural network architecture, we choose to build CNN model at first which has no people to approach it with our dataset. Then, we choose to build U-net model which is great in biology and medical domain. The reason why we want to use U-net is it can get precise localization with information.

Conclusion

We implemented CNN and U-net model to solve a salt identification problem. We achieved best accuracy at 0.9221 for U-net, and best accuracy at 0.8507 for CNN. Using linear regression, we find there has little relationship between depth and salt distribution. However, having just over 4000 images to train data for salt identification is still not enough to solve the geology problem. Also,

there has some limitations by using our computer. Testing more than 100 epochs, our computer will die even we really want it can be achieved.

Both of us are very satisfied with the results and we all think our efforts deserve it.

In future, we plan to implement more models and change more parameters to adjust. We even want to mix two different models together to check if it will boost original models. We also plan on collecting more training samples from seismic website and other sources, such that the model can be trained properly.

References

[1] https://en.wikipedia.org/wiki/Reflection_seismology

[2] https://www.aqua-calc.com/page/density-table/substance/salt

[3]https://www.iongeo.com/content/documents/Resource%20Center/Articles/INT_Imaging_Salt_tutorial_141101.pdf

[4] https://en.wikipedia.org/wiki/Convolutional_neural_network

[5] https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

[6] http://deeplearning.net/tutorial/unet.html

[7] https://towardsdatascience.com/medical-image-segmentation-part-1-unet-convolutional-networks-with-interactive-code-70f0f17f46c6

[8] http://cs231n.github.io/convolutional-networks/

[9] https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[10] https://en.wikipedia.org/wiki/Convolutional_neural_network

[11] https://en.wikipedia.org/wiki/Loss_function

[12]https://pdfs.semanticscholar.org/0704/5f87709d0b7b998794e9fa912c0aba912281.pdf

[13] https://www.depends-on-the-definition.com/unet-keras-segmenting-images/