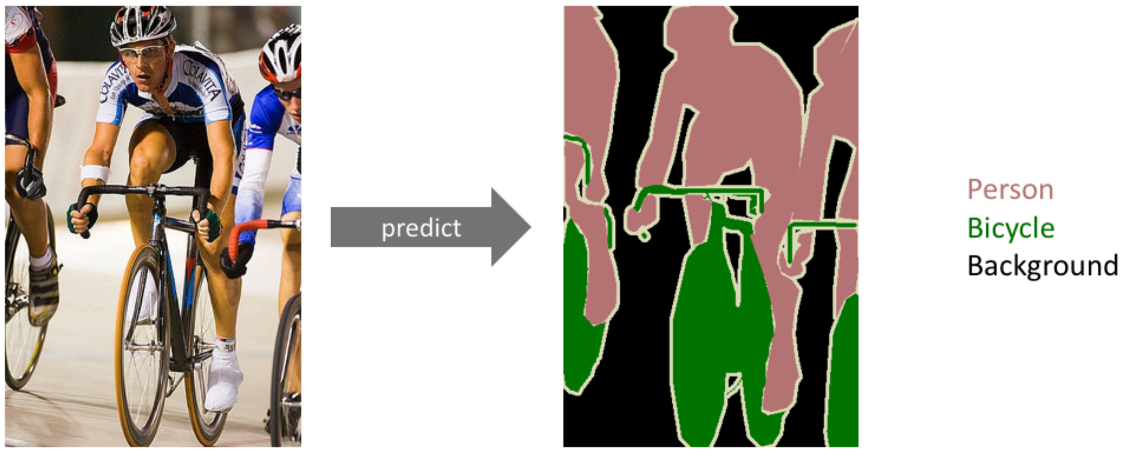
BE - Semantic Segmentation[1]

A Salt Identification Case Study

The goal of semantic image segmentation is to label each pixel of an image with a corresponding class of what is being represented. Because we're predicting for every pixel in the image, this task is commonly referred to as dense prediction.



Unlike the other computer vision tasks, e.g., image classification, object detection, , the expected output in semantic segmentation are not just labels and bounding box parameters. The output itself is a high resolution image (typically of the same size as input image) in which each pixel is classified to a particular class. Thus it is a pixel level image classification.

We will also consider a practical real world case study to understand the importance of semantic segmentation. The problem statement and the datasets are described in the below sections.

1. Business Problem

In any Machine Learning task, it is always suggested to spend a decent amount of time in aptly understanding the business problem that we aim to solve. This not only helps to apply the technical tools efficiently but also motivates the developer to use his/her skills in solving a real world problem.

[1] This BE has been adapted from the blog by Harshall Lamba: https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

TGS is one of the leading Geo-science and Data companies which uses seismic images and 3D renderings to understand which areas beneath the Earth's surface which contain large amounts of oil and gas.

Interestingly, the surfaces which contain oil and gas, also contain huge deposits of salt. So with the help of seismic technology, they try to predict which areas in the surface of the Earth contain huge amount of salts. Unfortunately, professional seismic imaging requires expert human vision to exactly identify salt bodies. This leads to highly subjective and variable renderings. Moreover it could cause huge loss for the oil and gas company drillers if the human prediction is incorrect. Thus TGS hosted a Kaggle Competition, to employ machine vision to solve this task with better efficiency and accuracy.

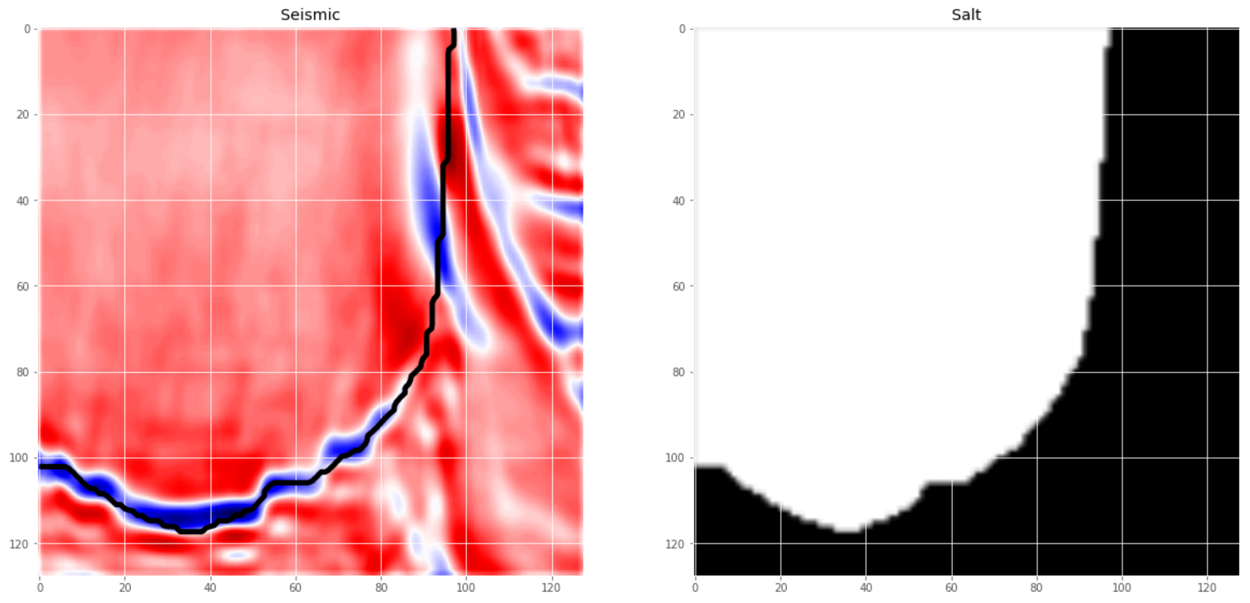
To read more about the challenge, click here.
To read more about seismic technology, click here.

2. Understanding the data

Download the data files from here.
For simplicity we will only use train.zip file which contains both the images and their corresponding masks.

In the images directory, there are 4000 seismic images which are used by human experts to predict whether there could be salt deposits in that region or not.

In the masks directory, there are 4000 gray scale images which are the actual ground truth values of the corresponding images which denote whether the seismic image contains salt deposits and if so where. These will be used for building a supervised learning model.
Let's visualize the given data to get a better understanding:

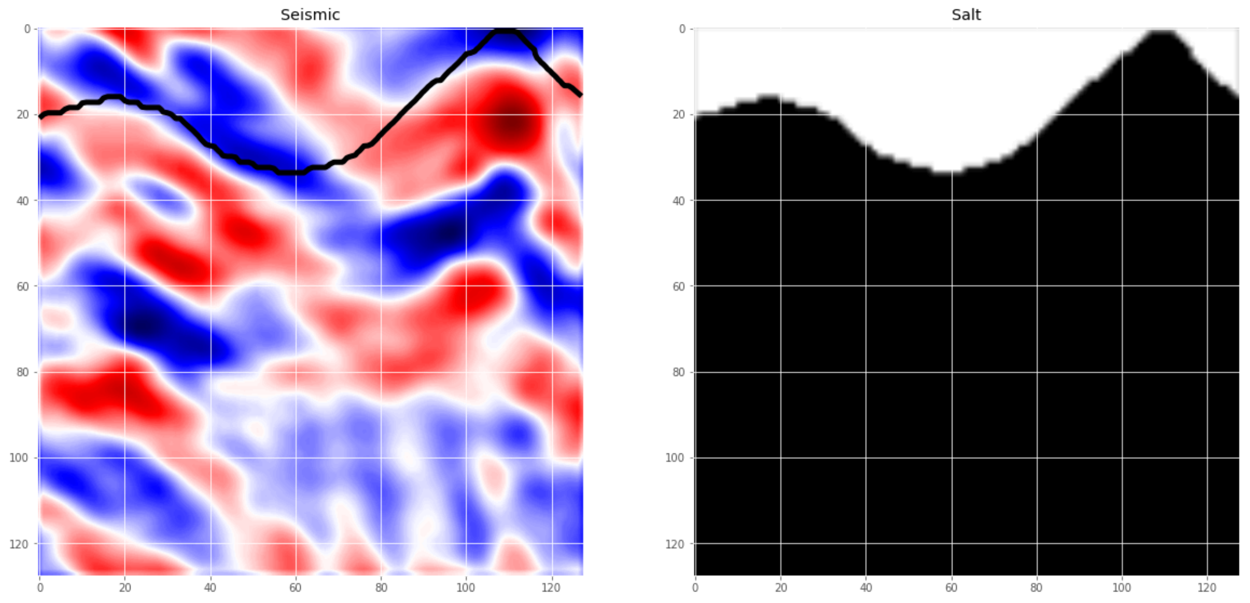


Sample data point and corresponding label

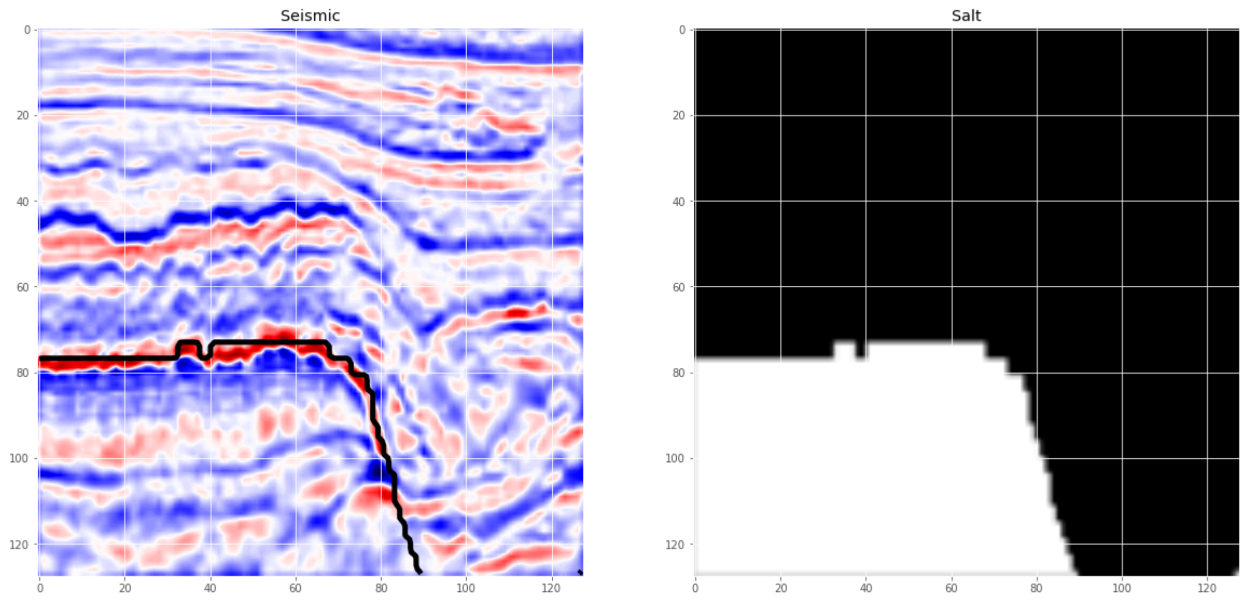
The image on left is the seismic image. The black boundary is drawn just for the sake of understanding denoting which part contains salt and which does not. (Of course this boundary is not a part of the original image)

The image on the right is called as the mask which is the ground truth label. This is what our model must predict for the given seismic image. The white region denotes salt deposits and the black region denotes no salt.

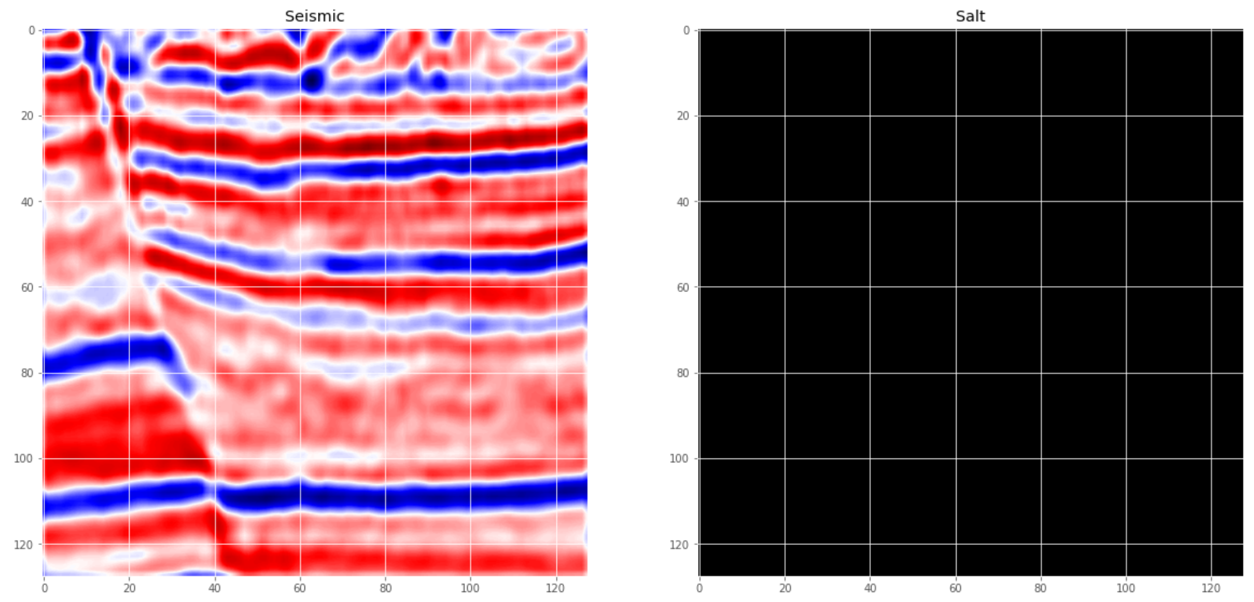
Let’s look at a few more images:



Sample data point and corresponding label



Sample data point and corresponding label



Sample data point and corresponding label

Notice that if the mask is entirely black, this means there are no salt deposits in the given seismic image. Clearly from the above few images it can be inferred that it's not easy for human experts to make accurate mask predictions for the seismic images.

3. UNET Architecture and Training

The UNET was developed by Olaf Ronneberger *et al.* for Bio Medical Image Segmentation. The architecture contains two paths. First path is the contraction path (also called as the encoder) which is used to capture the context in the image. The encoder is just a traditional stack of convolutional and max pooling layers. The second path is the symmetric expanding path (also called as the decoder) which is used to enable precise localization using transposed convolutions. Thus it is an end-to-end fully convolutional network (FCN), i.e. it only contains Convolutional layers and does not contain any Dense layer because of which it can accept image of any size.

In the original paper, the UNET is described as follows:

2

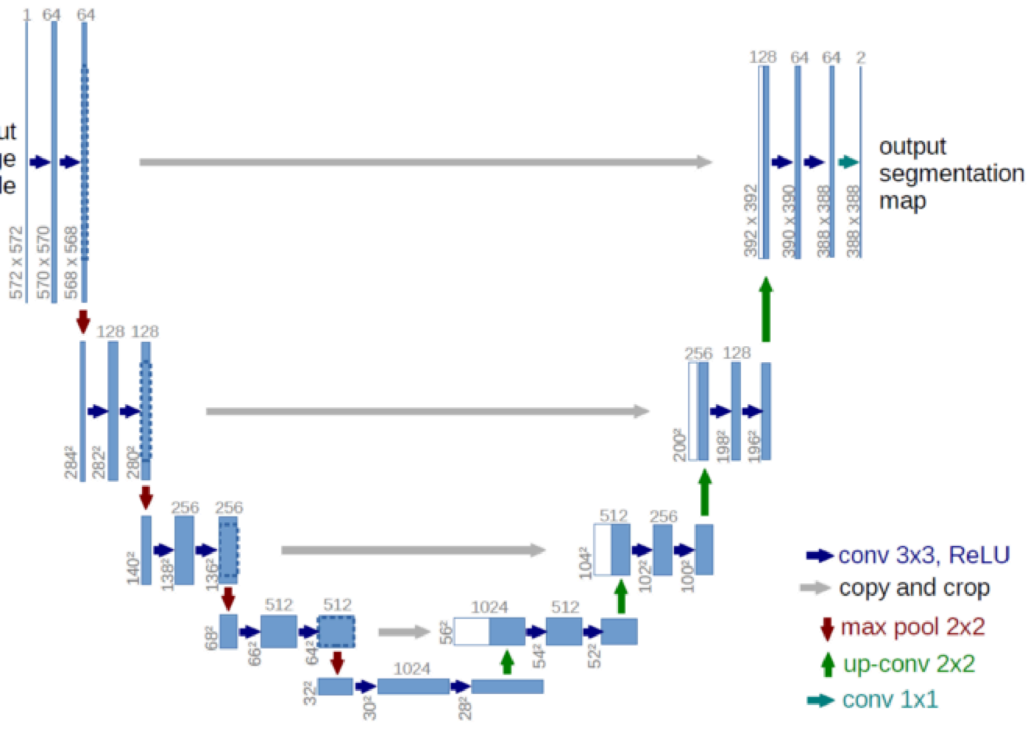
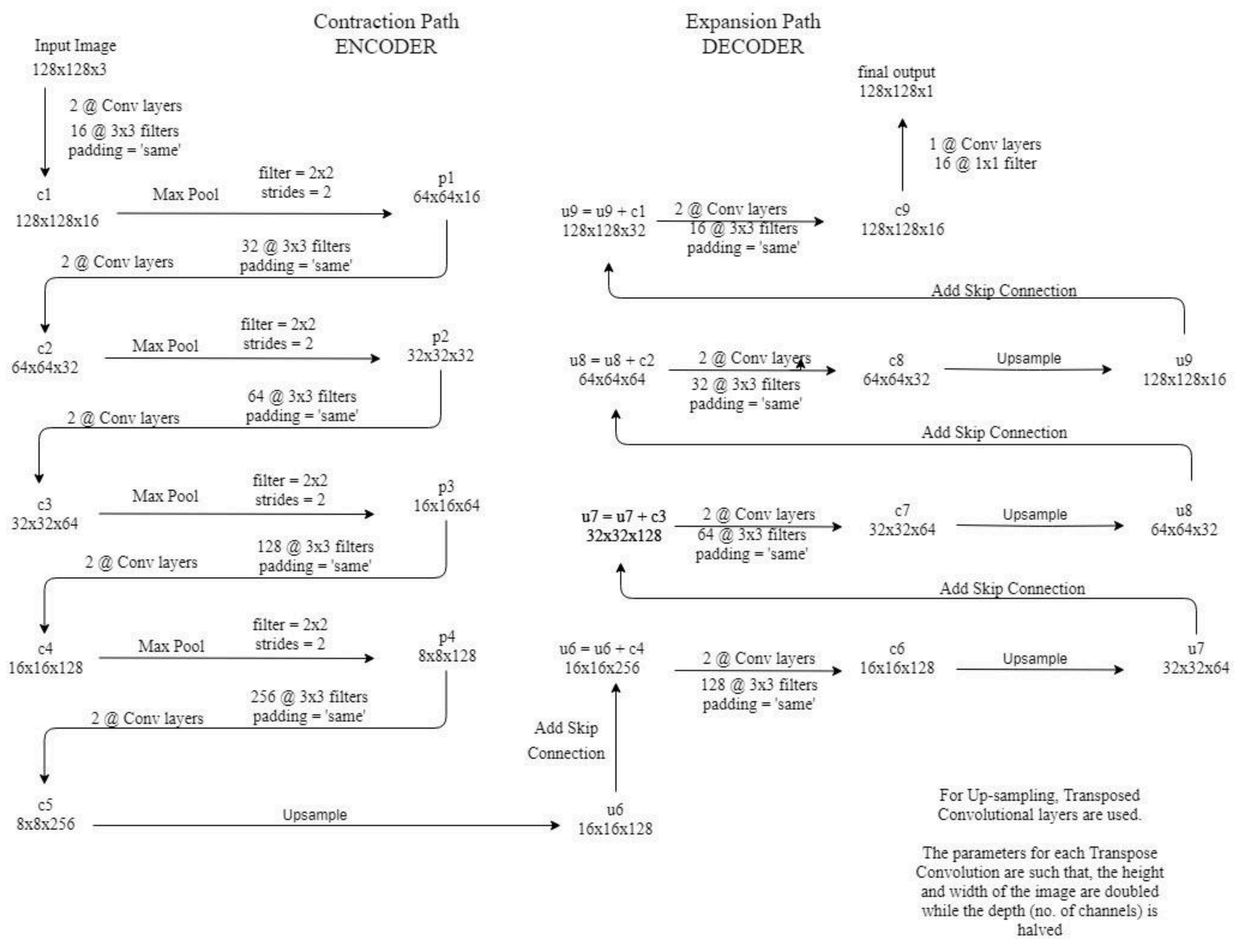


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Note that in the original paper, the size of the input image is 572x572x3, however, we will use input image of size 128x128x3. Hence the size at various locations will differ from that in the original paper but the core components remain the same.
Below is the detailed explanation of the architecture:



Detailed UNET Architecture

Points to note:

- 2@Conv layers means that two consecutive Convolution Layers are applied
- c1, c2, …. c9 are the output tensors of Convolutional Layers
- p1, p2, p3 and p4 are the output tensors of Max Pooling Layers
- u6, u7, u8 and u9 are the output tensors of up-sampling (transposed convolutional) layers
- The left hand side is the contraction path (Encoder) where we apply regular convolutions and max pooling layers.
- In the Encoder, the size of the image gradually reduces while the depth gradually increases. Starting from 128x128x3 to 8x8x256
- This basically means the network learns the “WHAT” information in the image, however it has lost the “WHERE” information
- The right hand side is the expansion path (Decoder) where we apply transposed convolutions along with regular convolutions

- In the decoder, the size of the image gradually increases and the depth gradually decreases. Starting from 8x8x256 to 128x128x1
- Intuitively, the Decoder recovers the “WHERE” information (precise localization) by gradually applying up-sampling
- To get better precise locations, at every step of the decoder we use skip connections by concatenating the output of the transposed convolution layers with the feature maps from the Encoder at the same level:
  u6 = u6 + c4
  u7 = u7 + c3
  u8 = u8 + c2
  u9 = u9 + c1
  After every concatenation we again apply two consecutive regular convolutions so that the model can learn to assemble a more precise output
- This is what gives the architecture a symmetric U-shape, hence the name UNET
- On a high level, we have the following relationship:
  Input (128x128x1) => Encoder =>(8x8x256) => Decoder =>Ouput (128x128x1)

Please write the Pythorch or Keras code to define the above model;

Training

Model is compiled with Adam optimizer and we use binary cross entropy loss function since there are only two classes (salt and no salt).

Keras callbacks could be used to implement:

- Learning rate decay if the validation loss does not improve for 5 continues epochs.
- Early stopping if the validation loss does not improve for 10 continues epochs.
- Save the weights only if there is improvement in validation loss.

Please use a batch size of 32.
Note that there could be a lot of scope to tune these hyper parameters and further improve the model performance.

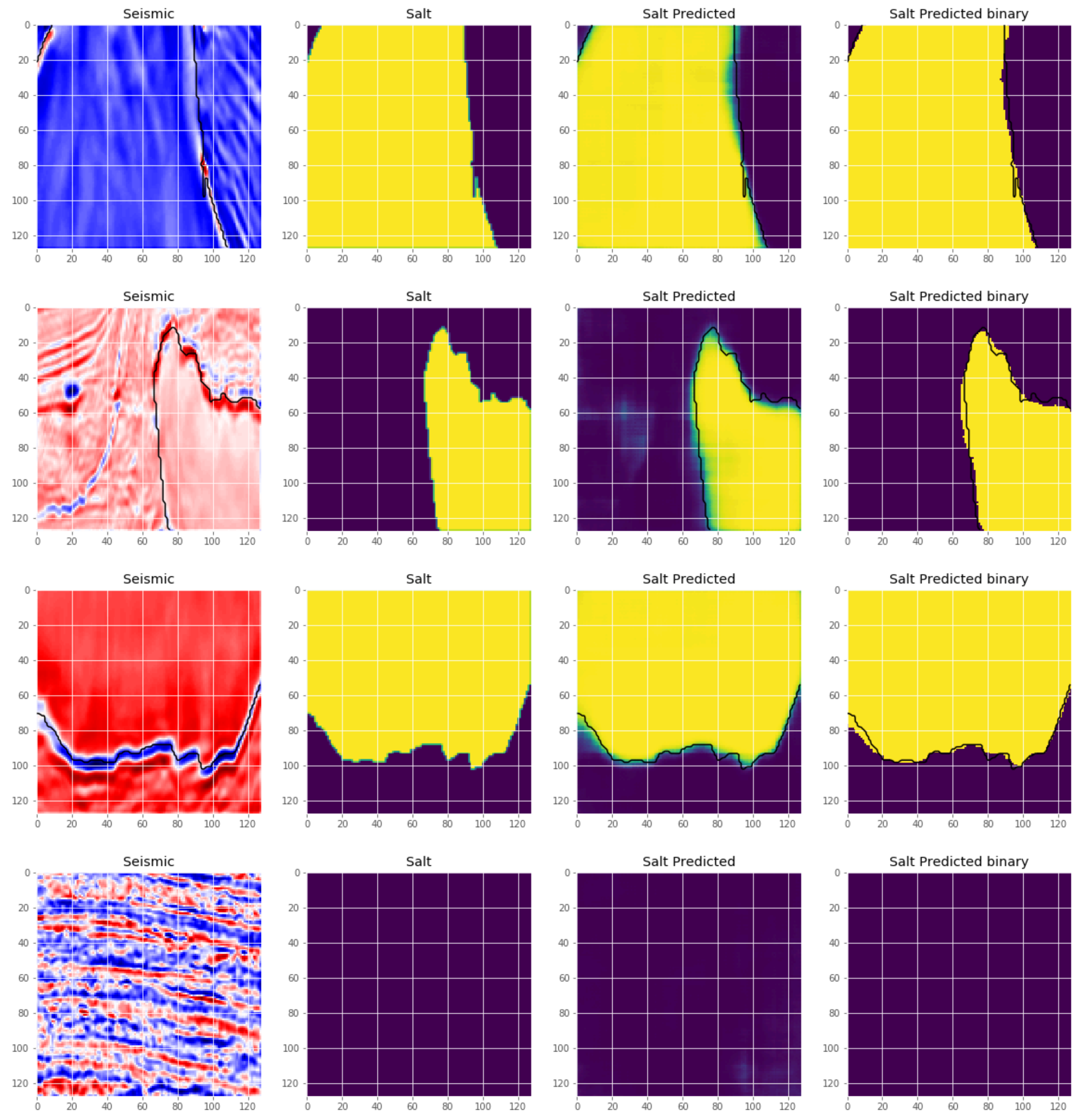
4. Inference

Note that for each pixel we get a value between 0 to 1. 0 represents no salt and 1 represents salt.

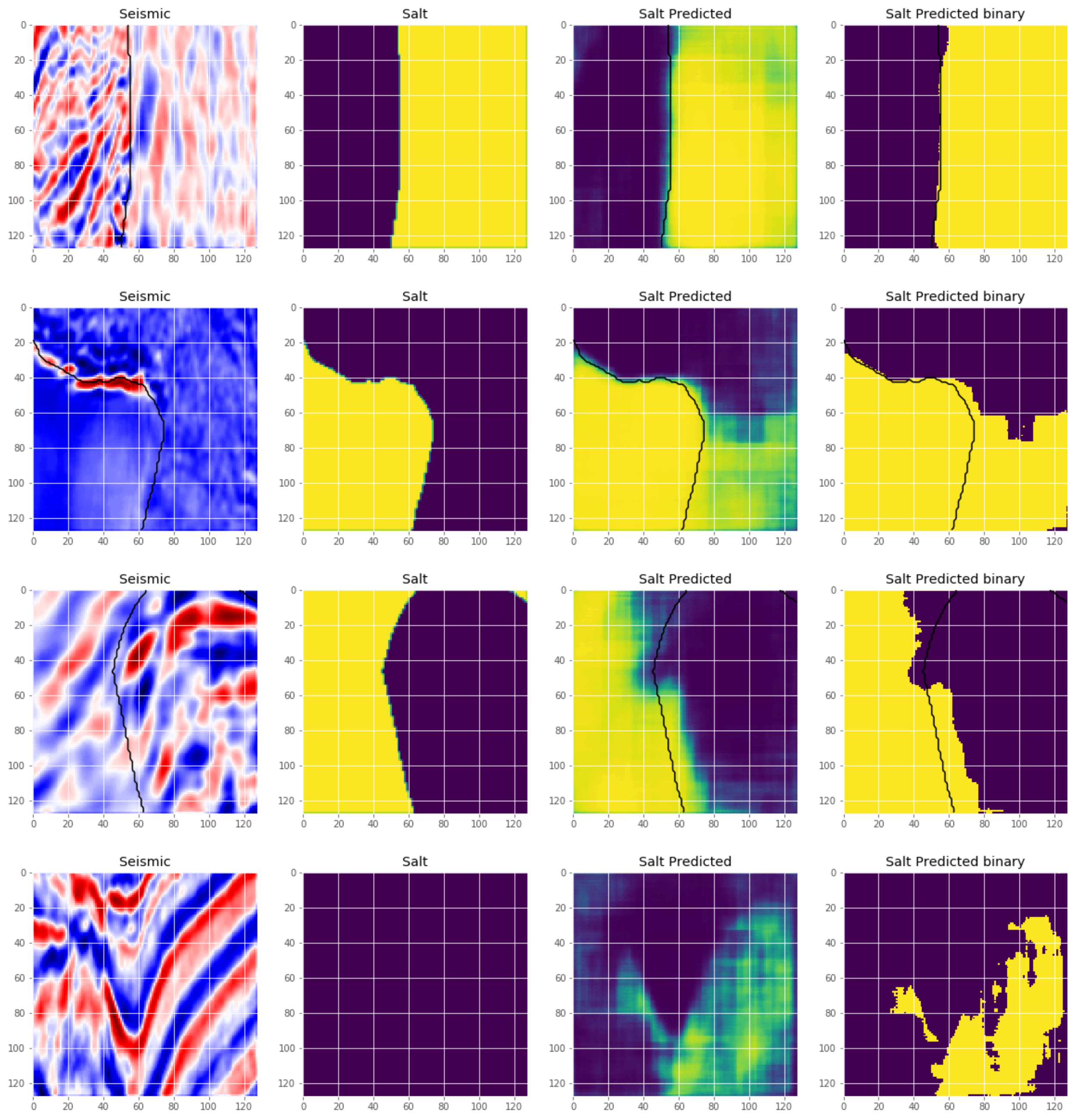
We take 0.5 as the threshold to decide whether to classify a pixel as 0 or 1. However deciding threshold is tricky and can be treated as another hyper parameter.

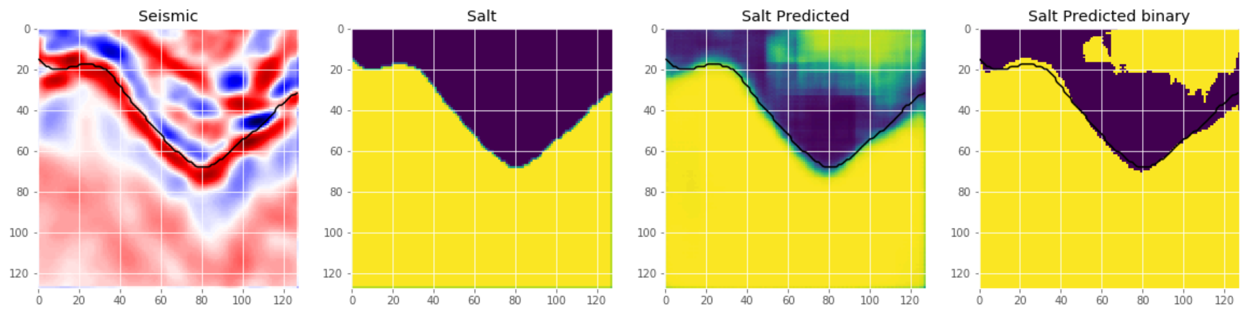
Let’s look at some results on both training set and validation set:

Results on training set



Results on Validation Set





Results on training set are relatively better than those on validation set which implies the model suffers from overfitting. One obvious reason could be the small number images used to train the model.

5. Questions

In this section we investigate the impact of various architectural choices of the proposed U-Net for semantic segmentation.

5.1 Max pooling

Max pooling has been used for down-sizing the input volume. Instead of max pooling, is it possible to simply make use of convolution with a different stride, e.g., 2 ? How does it impact the performance ?

5.2 Skip Connections

The skip connections have been used in the proposed architecture. Why is this important ? Suppose that we remove these skip connections from the proposed architecture, how does it impact the result ? Please explain your intuition. Furthermore, the current skip connections consist of concatenating feature maps of the encoder with those of the decoder at the same level, is it possible to make use of other alternatives, for instance addition, Max or Min operation ? Do they impact the results ? Explain.

5.3 FCN and Auto-encoder

The proposed architecture is an auto-encoder like architecture, i.e., composed of an encoder and a decoder. Is it necessary ? Is it possible that we make use of a fully convolutional neural network (FCN) consisting of series of convolutions from the input to the output while keeping the image size, i.e., width and height, during all the convolutions ?

5.4 Threshold for inference

We have taken 0.5 as the threshold to decide whether to classify a pixel as salt or not. However, this choice proves not to be the optimal in terms of precision and recall rate for pixels being classified salt. Could you explain how we should define the precision and recall rate in our business case. Could you propose a strategy to improve both precision and recall ?

6. References

- https://arxiv.org/abs/1505.04597
- https://www.depends-on-the-definition.com/unet-keras-segmenting-images/
- https://towardsdatascience.com/up-sampling-with-transposed-convolution-9ae4f2df52d0
- Transposed convolution: Up sampling with Transposed Convolution