ABSTRACT

Seismic-data interpretation has the main goal of the identification of compartments, faults, fault sealing, and trapping mechanism that hold hydrocarbons; it additionally tries to understand the depositional history of the environment to describe the relationship between seismic data and a priori geological information. Imaging salt has been a huge topic in the seismic industry, basically since they imaged salt the first time. Salt bodies are important for the hydrocarbon industry, as they usually form nice oil traps. So there's a clear motivation to delineate salt bodies in the subsurface. Seismic data interpreters are used to interpret 2D or 3D images that have been heavily processed. In our problem statement, we are dealing with data that is less noisy which is an added advantage. Our solution to the problem is to basically use U-Nets which have shown state of the art results on image segmentation. Each pixel in the image is checked for the presence of salt and further on this is how the proportion of salt in that seismic image is calculated. The energy function is computed by a pixel-wise soft-max over the final feature map combined with the binary cross entropy loss function. Our model boasts an accuracy of 94.7% which is significantly higher than that of in the previous implementations.

ACKNOWLEDGMENT

Any achievement does not depend solely on the individual efforts but on the guidance, encouragement and co-operation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped us in carrying out this project work. We would like to take this opportunity to thank them all.

We would like to express my profound thanks to **Sri. G Dayanand,** Chairman, Sapthagiri College of Engineering Bangalore, for their continuous support in providing amenities to carry out this Project.

Special Thanks to **Dr. N. Srinivasan,** Director, Sapthagiri College of Engineering Bangalore, for his valuable suggestion.

Also we would like to express our immense gratitude to **Dr. K L SHIVABASAPPA** Principal, Sapthagiri College of Engineering Bangalore, for their help and inspiration during the tenure of the course.

We also extend our sincere thanks to **Dr. Kamalakshi Naganna**, Professor and Head (IC), Department of Computer Science and Engineering, Sapthagiri College of Engineering, for their constant support.

We would like to express our heartful gratitude to **Mrs. Madhushree, Asst. Professor**, Department of Computer Science and Engineering, Sapthagiri College of Engineering, for their timely advice on the project and regular assistance throughout the work.

We also extend our sincere thanks to all the **Faculty members** and **Supporting staff** Department of Computer Science and Engineering, Sapthagiri College of Engineering, for their constant support and encouragement.

Finally, we thank our parents and friends for their moral support.

ABHISHEK VARMA **1SG15CS004**
AMAN GUPTA **1SG15CS008**
DYLAN SALDANHA **1SG15CS026**

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Background

Hydrocarbon exploration (or oil and gas exploration) is the search by petroleum geologists and geophysicists for deposits of hydrocarbons, particularly petroleum and natural gas, in the Earth using petroleum geology.

Visible surface features such as oil seeps, natural gas seeps, pockmarks (underwater craters caused by escaping gas) provide basic evidence of hydrocarbon generation (be it shallow or deep in the Earth). However, most exploration depends on highly sophisticated technology to detect and determine the extent of these deposits using exploration geophysics. Areas thought to contain hydrocarbons are initially subjected to a gravity survey, magnetic survey, passive seismic or regional seismic reflection surveys to detect large-scale features of the sub-surface geology. Features of interest (known as leads) are subjected to more detailed seismic surveys which work on the principle of the time it takes for reflected sound waves to travel through matter (rock) of varying densities and using the process of depth conversion to create a profile of the substructure. Finally, when a prospect has been identified and evaluated and passes the oil company's selection criteria, an exploration well is drilled in an attempt to conclusively determine the presence or absence of oil or gas. Offshore the risk can be reduced by using electromagnetic methods.

Oil exploration is an expensive, high-risk operation. Offshore and remote area exploration is generally only undertaken by very large corporations or national governments. Typical shallow shelf oil wells (e.g. North Sea) cost US$10 – 30 million, while deep water wells can cost up to US$100 million plus. Hundreds of smaller companies search for onshore hydrocarbon deposits worldwide, with some wells costing as little as US$100,000.

Hydrocarbon exploration is a high risk investment and risk assessment is paramount for successful project portfolio management. Exploration risk is a difficult concept and is usually defined by assigning confidence to the presence of the imperative geological factors, as discussed above. This confidence is based on data and/or models and is usually mapped on Common Risk Segment Maps (CRS Maps). High confidence in the presence of imperative geological factors is usually coloured green and low confidence coloured red. Therefore, these maps are also called Traffic Light Maps (Figure 1.1), while the full procedure is often referred to as Play Fairway Analysis. The aim of such procedures is to force the geologist to objectively assess all different geological factors. Furthermore, it

results in simple maps that can be understood by non-geologists and managers to base exploration decisions on.

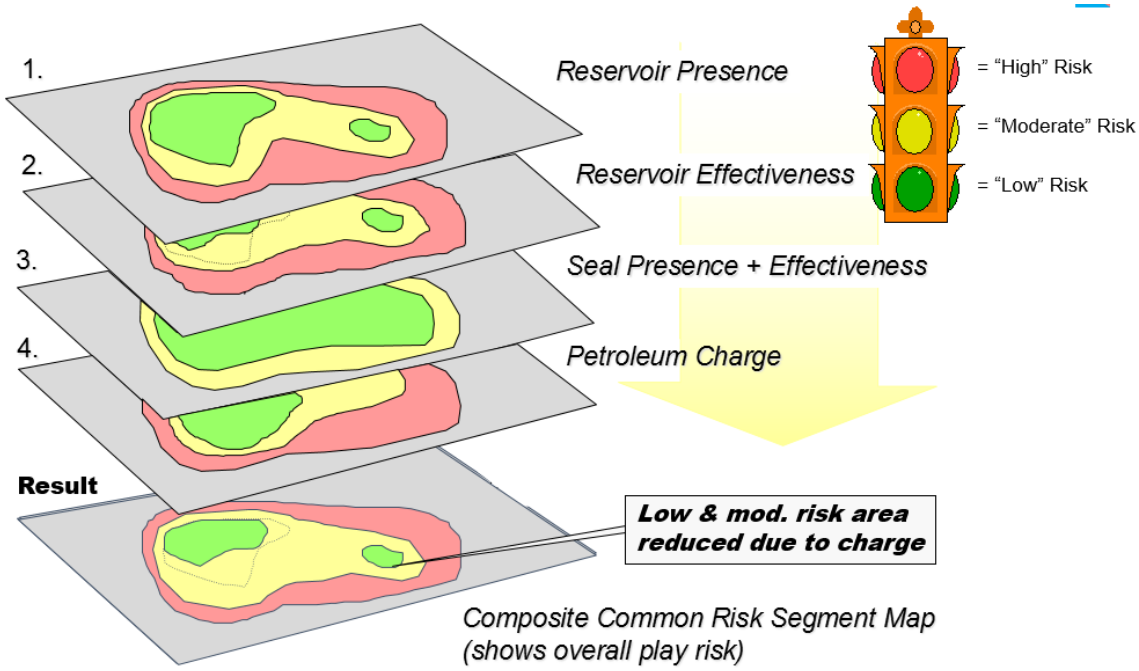


Figure 1.1.0.1 Traffic Light Maps

A prospect is a potential trap which geologists believe may contain hydrocarbons. A significant amount of geological, structural and seismic investigation must first be completed to redefine the potential hydrocarbon drill location from a lead to a prospect. Four geological factors have to be present for a prospect to work and if any of them fail neither oil nor gas will be present.

a. Source rock

When organic-rich rock such as oil shale or coal is subjected to high pressure and temperature over an extended period of time, hydrocarbons form.

b. Migration

The hydrocarbons are expelled from source rock by three density-related mechanisms: the newly matured hydrocarbons are less dense than their precursors, which causes over-pressure; the hydrocarbons are lighter, and so migrate upwards due to buoyancy, and the fluids expand as further burial causes increased heating. Most hydrocarbons migrate to the surface as oil seeps, but some will get trapped.

c. Reservoir

The hydrocarbons are contained in a reservoir rock. This is commonly a porous sandstone or limestone. The oil collects in the pores within the rock although open

fractures within non-porous rocks (e.g. fractured granite) may also store hydrocarbons. The reservoir must also be permeable so that the hydrocarbons will flow to surface during production.

d. Trap

The hydrocarbons are buoyant and have to be trapped within a structural (e.g. Anticline, fault block) or stratigraphic trap. The hydrocarbon trap has to be covered by an impermeable rock known as a seal or cap-rock in order to prevent hydrocarbons escaping to the surface

As per the oil market report of International Energy Agency, global oil demand is estimated to increase by 1.5 million b/d to 99.3 million b/d in 2018. Scientific observation says that there are around 5,500 varieties of naturally occurring minerals on earth. Do we know about all? Is there any shortcut to find out oil and gas depositions? Exploring oil and gas under land or within the seabed using surface methods is based on two main principles. One is to survey geological features of the land to determine sedimentary rock formation, repeated folds, and faults. The other is to identify the hydrocarbon seepage on the earth surface.

Using AI in deep-sea exploration may not give instant results but it attempts to give a boost to the detection know-hows. For example, if a cop wants to catch hold of criminals they first attempt to track their voice and text transactions with kith and kin. They attempt to catch hold of their close ones to trap the criminal. Similarly, Artificial Intelligence attempts to identify natural seeps that occur when oil escapes from the fractured rocks found in ocean bed or on the ground. These seeps become tar-like deposits which are the best indicators to find energy sources.

A trained computer vision model can identify the indicators of energy sources. The efficiency of the model depends on the data that is annotated and fed for system learning. By annotating with polygon tool irregular object shape detection is possible accurately, which helps in training models learn and detect the indicators of energy sources. OCLAVI is a platform, where the client can annotate image objects precisely even with a low-quality image data sets with help of various tools like box annotation tool, cuboid, polygon, point and circle annotation tools.

Here are 7 operational excellence challenges facing the oil and gas industry.

Challenge #1: Increasing visibility into complex operations to control costs and optimize the performance of employees, facilities and assets.

Oil and gas companies operate in some of the most physically and politically challenging environments on earth. Add to that factors such as volatile market prices, fluctuating demand, complex compliance and regulatory regimes, projects that involve multiple third

party suppliers, and a workforce that has widely varying education and skill levels — to name but a few — and it's clear that oil and gas companies have some of the most complex operations on earth.

In order to manage risks, control costs and optimize the performance of employees, facilitate and assets, oil and gas companies need to gain increased visibility into their operations. "One strategy for achieving this has been the adoption of a 'digital oilfield' or 'integrated operations' to enhance reservoir recoverability, optimize production, and reduce economic, environment, health, and safety risks," explains IDC analyst Roberta Bigliani. "Initially this strategy was only associated with upstream, but companies are increasingly focused on accessing and managing key asset-related data to improve decision making across the entire enterprise from field to refinery."

Challenge #2: Improve collaboration with oilfield services to improve logistics. Even vertically integrated oil giants like Chevron and ExxonMobil must rely on third party suppliers to provide specialist equipment and expertise for different parts of the oil and gas supply chain. The complexity of these arrangements became part of public consciousness following BP's Deepwater Horizon rig disaster as investigators debated liability in the disaster given the role of several companies — including BP, Anadarko, and Halliburton — in the operation of the rig. For oil and gas companies, oilfield services perform business critical functions. Thus, a mistake or inefficiency caused by one company can have a devastating knock on effect. Additionally, poor collaboration and communication can slow down projects and increase inefficiency.

"E&P companies expect a lot from their service providers and OFS companies want to deliver," observes an Accenture whitepaper. "However, poor material planning — exacerbated by market volatility and supply limitations — often dampens speed and productivity."

Challenge #3: Develop a high-performing culture through training, new systems and ongoing management. Employee onboarding, retention and training have become a critical issue in oil and gas as competition for talent heats up. Emerging markets such as India and China are starting to open up — not only increasing global demand for oil and gas but also competing for talent. BP's Energy Outlook 2035, for instance, expects the global energy demand will rise by 41 per cent from 2012 to 2035 with the vast majority (95%) of that coming from emerging economies. "We have seen an increase in the number of projects worldwide, not just in the emerging markets, but overall capital projects and the type of projects, the complexity of projects is really putting the system under pressure," observes

Sheryl-Ann Carscadden, Director of Services at Calgary-based consulting firm Ethier in an interview.

The talent shortage isn't helped by the looming retirement of the industry's most experienced workers; the Society of Petroleum Engineers estimates that up to 50% of skilled workers could retire from the oil and gas industry within the next five to seven years. Usage of gas in the primary energy mix is likely to increase from 22% to 25%. Shaping the world energy system with AI revolution brings an increase in global GDP by 2030

1.2 Overview of Present Work

Many studies and research have been conducted on emotion recognition from voice, a majority of which lack accuracy, and the usage of varied techniques. In [1], for hydrocarbon exploration, large volumes of data are acquired and used in physical modeling-based workflows to identify geologic features of interest such as fault networks, salt bodies, or, in general, elements of petroleum systems. The adjoint modeling step, which transforms the data into the model space, and subsequent interpretation can be very expensive, both in terms of computing resources and domain-expert time. We propose and implement a unique approach that bypasses these demanding steps, directly assisting interpretation. This is done by training a deep neural network to learn a mapping relationship between the data space and the final output (particularly, spatial points indicating fault presence). The key to obtaining accurate predictions is the use of the Wasserstein loss function, which properly handles the structured output — in our case, by exploiting fault surface continuity. The promising results shown here for synthetic data demonstrate a new way of using seismic data and suggest more direct methods to identify key elements in the subsurface.

In [2], The authors apply deep learning techniques to the problem of the salt body detection in seismic images. They consider salt body classification as an image segmentation problem, and propose to design a multi-layer convolutional neural network, feed in training data to train this network, and test the model using blind test data. Their results indicate that the proposed network architecture and workflow are capable of capturing subtle salt features automatically without the need for manual input. Trained with a limited amount of inline sections, the model can generalize to the blind test data and be efficiently applied to a whole 3D volume of seismic data.

In [3] a novel approach to detect salt bodies based on seismic attributes and supervised learning. They report on the use of a machine learning algorithm, Extremely Randomized Trees, to automatically identify and classify salt regions. They have worked with a complex

synthetic seismic dataset from phase I model of the SEG Advanced Modeling Corporation (SEAM) that corresponds to deep water regions of the Gulf of Mexico. This dataset has very low frequency and contains sediments bearing amplitude values similar to those of salt bodies. In the first step of our methodology, where machine learning is applied directly to the seismic data, they obtained accuracy values of around 80%. A second (post-processing) smoothing step improved accuracy to around 95%. They conclude that machine learning is a promising mechanism to identify salt bodies on seismic data, especially with models that can produce complex decision boundaries, while being able to control the associated variance component of error.

[4] With recent advances in machine learning, convolutional neural networks (CNNs) have been successfully applied in many fields, and several attempts have been made in the field of geophysics. In this letter, they investigated the mapping of subsurface electrical resistivity distributions from electromagnetic (EM) data with CNNs. To begin imaging electrical resistivity using CNNs, they carried out precise delineation of a subsurface salt structure, which is indispensable for identification of offshore hydrocarbon reservoirs, using towed streamer EM data. For training the CNN model, an electrical resistivity model, including a salt body, and corresponding EM data calculated through numerical modeling were used as the label and input, respectively. The optimal weights and biases of the CNN were obtained minimizing the mean-square error between the predicted resistivity distribution and the target label. The final CNN model was selected using a validation data set during training. After training, they applied the trained CNN to test data sets of noisy data and simulated-SEAM data, which were not provided to the network during training. The test results demonstrate that our trained CNN model is stable, reliable, and efficient, and indicate the possibility of successful application of our CNN model to field data. Our study has shown the promising potential of CNNs for identifying defined subsurface electrical resistivity structures that are difficult to find using conventional EM inversion.

[5] The exploration of petroleum reservoirs has a close relationship with the identification of salt domes. To efficiently interpret salt-dome structures, in this paper, they propose a method that tracks salt-dome boundaries through seismic volumes using a tensor-based subspace learning algorithm. They build texture tensors by classifying image patches acquired along the boundary regions of seismic sections and contrast maps. With features extracted from the subspaces of texture tensors, we can identify tracked points in neighboring sections and label salt-dome boundaries by optimally connecting these points.

Experimental results show that the proposed method outperforms the state-of-the-art salt-dome detection method by employing texture information and tensor-based analysis.

1.3 Problem Statement

To find potential reserves of hydrocarbons by identifying salt bodies in the vicinity of the area under inspection.

Several areas of Earth with large accumulations of oil and gas also have huge deposits of salt below the surface. But unfortunately, knowing where large salt deposits are precisely is very difficult. Professional seismic imaging still requires expert human interpretation of salt bodies. This leads to very subjective, highly variable renderings. More alarmingly, it leads to potentially dangerous situations for oil and gas company drillers.

1.4 Objectives

- Our main objective is to identify such potential segments of land that contain significant amount of salt based on the images of the land segments provided as input.
- Oil and Gas Companies can benefit from this by wisely choosing the segments of the land that contains more amount of Oil deposits, thereby making effective and rational use of company resources.
- To eliminate human intervention in the hydrocarbon exploration pipeline.
- Judicious use of Oil and Gas company resources in finding such potential sources.
- Reduce to risk involved to human lives involved in the process.

1.5 Organization of Project Report

This report deals with the implementation of the project ‘Hydrocarbon Exploration using Seismic Images”. This report is organized as 7 chapters, namely, introduction, Literature Review, System Requirements, System Designs / Methodology, implementation, Results and Discussion and lastly conclusion and future work.

Chapter 1 gives a brief introduction about the study of hydrocarbon exploration and how exploration of hydrocarbons is traditionally done, the challenges encountered during this process and the impact on the economy. It consists of description of background, overview of the present work, problem statement and objectives.

Chapter 2 deals with the detailed analysis of previous studies and research conducted. It consists of summary of prior works, outcome of the review- problems identified, and proposed works.

Chapter 3 specifies the System Requirements. It consists of specification of all functional, and non-functional requirements along with the software/ hardware that has been used in the project.

Chapter 4 specifies the design details. Design is the process of establishing a system that will satisfy the previously identified functional and non- functional requirements. It also consists of specification of the architecture of the system and the major algorithm incorporated.

Chapter 5 includes the implementation part. Implementation is the process of converting the system design into an operational one. This phase starts after the completion of the development phase and must be carefully planned and controlled as it is a key stage. It includes a list of main packages, some of the user- defined functions and data structures.

Chapter 6 includes the description and the discussion of the results obtained. It consists of Testing and Results. Testing includes the Testing part which is an investigation conducted to provide stakeholders with information of the quality of the product or service under test. It also gives a business an opportunity to understand the risks of software implementation. Test techniques include, but are not limited to the process of executing a program or application with the intent of finding software bugs.

Chapter 7 mentions the conclusion and future enhancements for the project. IT contains the summary of the work carried out, contributions and utility. Also mentioned are the glossary, acronyms, and the bibliography.

CHAPTER 2

LITERATURE REVIEW

2.1 Summary of prior works

2.1.1 Supervised learning to detect salt body

[1] Seismic-data interpretation has as its main goal the identification of compartments, faults, fault sealing, and trapping mechanism that hold hydrocarbons; it additionally tries to understand the depositional history of the environment to describe the relationship between seismic data and a priori geological information. Data mining or knowledge discovery in databases (KDD) has become a significant area both in academia and industry. Data mining is the process of extracting novel, useful and understandable patterns from a large collection of data.

A major strategy in this field is to apply data mining algorithms (Hastie, 2011) to classify points or parts of the 3D seismic data to reinforce correct data interpretations. Multiple studies have shown the benefits of using data mining techniques for seismic-data interpretation. For example, previous work has shown how to generate a set of seismic traces from velocity models containing faults with varying locality, using machine learning to identify the presence of a fault in previously unseen traces (Zhang et. al., 2014). Other techniques segment a seismic image into structural and stratigraphic geologic units (Hale, 2002), which is best done using global optimization methods (Shi et. al., 2000; Hale et. al., 2003). Another solution is to use unsupervised learning techniques (Coléou et. al., 2003), often relying on the application of Self Organizing Maps (Castro de Matos et. al., 2007). Our new approach is essentially a novel salt body detection workflow. The workflow as a whole envisions the creation of a software solution that can automatically identify, classify and delineate salt bodies from seismic data using seismic attributes and supervised learning algorithms.

Automated classification of salt bodies using machine learning. The approach aims at automatically identifying and delineating geological elements from seismic data. Specifically, we focus on the automatic classification of salt bodies using supervised learning techniques. In supervised learning we assume each element of study is represented as an n-component vector-valued random variable (X1 , X2 ,..,Xn), where each Xi represents an attribute or feature; the space of all possible feature vectors is called the input space X. We also consider a set {w1, w2,...,wk} corresponding to the possible classes; this

forms the output space W. A classifier or learning algorithm typically receives as input a set of training examples from a source domain, T = {(xi, wi)}, where x = (x1 , x2 ,…,xn) is a vector in the input space, and w is a value in the (discrete) output space.

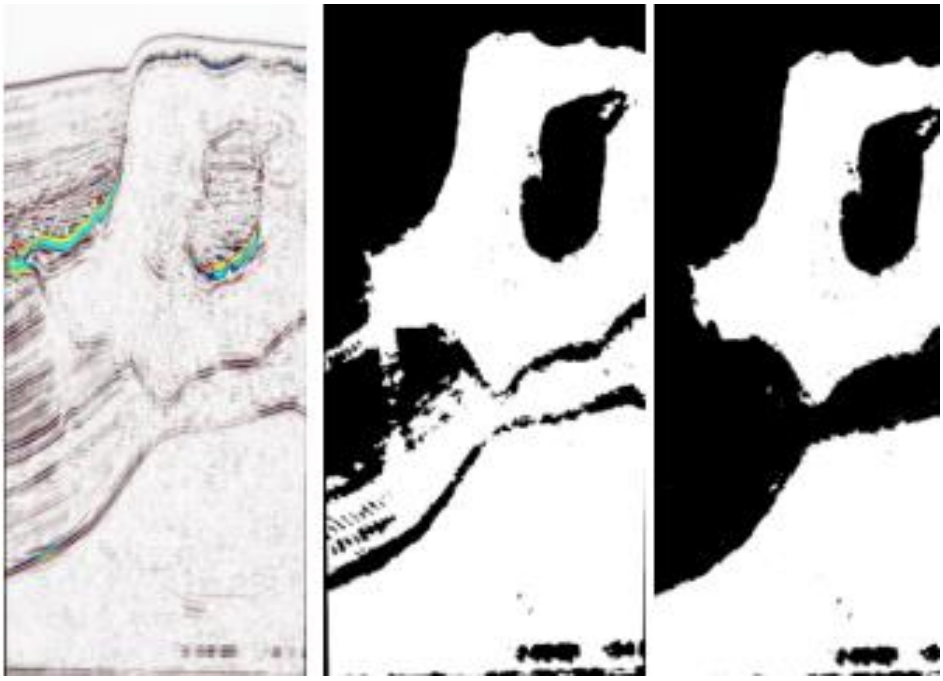


Figure 2.1.1.1 (a) Seismic Data (b)Classified Image (c)Results after post-processing step

They assume the training or source sample T consists of independently and identically distributed (i.i.d.) examples obtained according to a fixed but unknown joint probability distribution, P(x,w), in the input-output space. The outcome of the classifier is a hypothesis or function f(x) mapping the input space to the output space, f: X → W. We commonly choose the hypothesis that minimizes the expected value of a loss function (e.g., zero-one loss).

Informative attributes to generate predictive models in seismic data. A proper characterization of voxels can be attained with useful and informative features. We selected three features for our study exhibiting high correlation with the target class: signal amplitude (directly from seismic data), second derivative, and curve length; the last two derived from amplitude. Second derivative is instrumental to detect edges in images, and curve length capture patterns which characterize different features observed inside a salt structure and in its surroundings.

Supervised learning algorithms. Our data analysis phase receives as input a body of seismic data with the task of automatically identifying salt regions. To achieve a class-balanced problem, we made sure exactly one half of the subset corresponded to salt, and the other half as non-salt (the task exhibited equal class priors). The model was built using 2 million training voxels. Accuracy is estimated using 10-fold cross validation (Hastie,

2011). The classification model was subsequently used to automatically label the entire body of seismic data (376,752,501 voxels). Top performing learning algorithms were the following: Gradient Boosting Trees (Accuracy 80%), Extremely Randomized Trees (Accuracy 80%), and Random Forests (Accuracy 79%). All our learning algorithms are ensemble methods; these techniques have shown remarkable performance due to their ability to attain low bias (using complex decision boundaries), and low variance (achieved by averaging over various models).

SEAM I (SEG Advance Modeling Corporation) data, this comes from marine acquisition and represents strong challenges to the geophysical community. Inspiration was deep water (600 – 2000 meters) US GOM Salt Structure and its major structural features are salt body with rugose top and overhangs, twelves radial faults near the root salt, overturned sediment raft proximate to salt root and internal sutures and a heterogeneous salt cap. The migrated seismic volume was obtained with very low frequency, and there are sediments locations with similar amplitude value than salt body. Our final predictive model of choice was Extremely Randomized Trees, which was used to predict the labels of 376,752,501 samples; this resulted in a Boolean mask. The accuracy reported was essentially the same as in cross validation (80%). After that, we have removed outliers and misclassification using mathematical morphological operations and a 3D interactive guided (manual intervention) tool developed in house; finally, we used threshold segmentation using local average threshold to get better detection results.

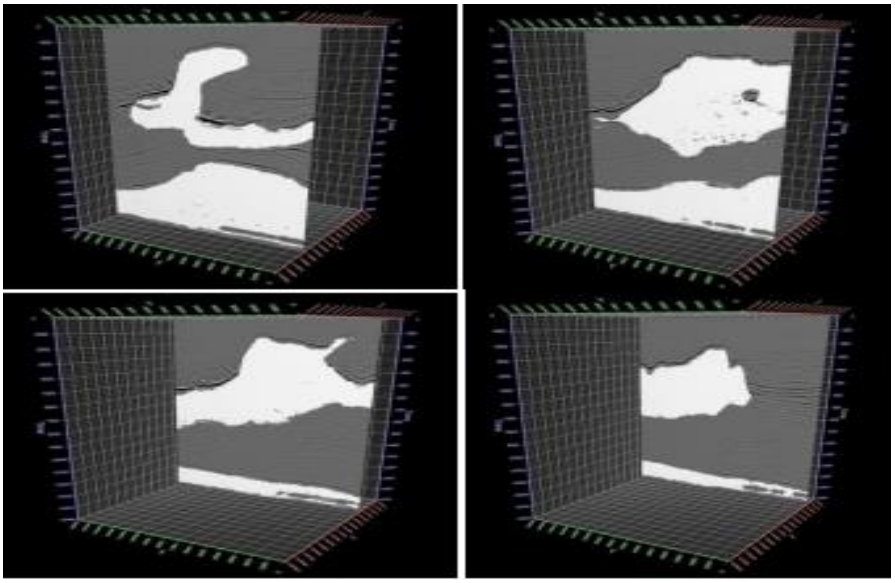


Figure 2.1.1.2: Overlapping between seismic data and salt body detected

To measure accuracy, we count the number of hits between the detected salt body and the interpretation in the following way: using both volumes, we have counted the number of

hits voxel by voxel. We refer to this number as NH. The effectiveness ratio is calculated as: (NH/TS) * 100, where TS is the total number of voxels in the volume. Following this technique, we have obtained an accuracy of 95.22%.

Results show very high accuracy when machine learning algorithms are used to predict class labels of voxels on a seismic cube; this is true even after training with a very small portion of the data (0.5%). After a first step, where machine learning is applied directly to the data, we obtained accuracy values of around 80%. A second (post-processing) step increased accuracy to around 95%. We conclude that machine learning is a promising mechanism to identify geological bodies on seismic data when the selected model has high capacity, and is able to control the variance component of error by model averaging (using ensemble techniques).

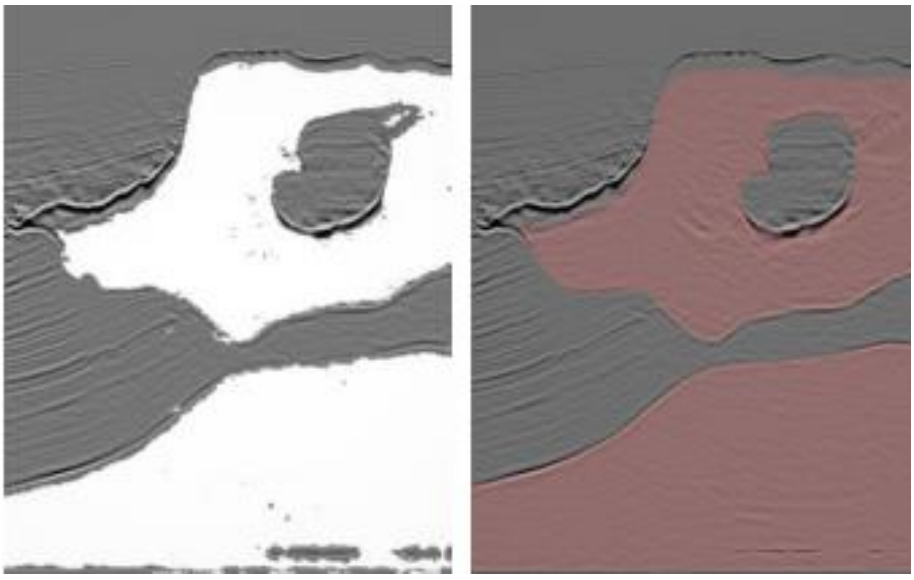


Figure 2.1.1.3: Overlapping between seismic data (a) salt body data (b) interpretation

2.1.2 Automated Fault Detection Without Seismic Processing

Recent work [4] demonstrates a new approach that builds and uses a deep neural network (DNN) statistical model to transform raw-input seismic data directly to the final mapping of faults in 2D. In this case, fault locations were chosen as the output because of their relevance to optimizing production in existing fields. DNNs are built on the premise that they can be used to replicate any function. This paper shows that DNNs can be used to identify fault structure in 3D volumes with reasonable accuracy.

The first step in Reference 5's workflow (Figure 4) is to collect the training examples. Real data examples are impractical and limiting since the labels are assigned to fault locations by few domain experts. This means that the neural network’s best result would be bound

by human performance and data quality. Instead, the idea here is to generate realistic 3D velocity models synthetically, with the fault labeling generated concurrently for an unbiased ground truth. Next, they use an acoustic approximation to the wave equation to generate wave fields and record them as time-series signals with predefined acquisition geometry. This step is conducted on thousands of random velocity-model realizations, giving many instances of labeled fault locations and the corresponding seismic traces for the entire data set. A portion of this data set is kept unseen from the algorithm (holdout set) so that it can be used for testing after training the predictor. Ranges have been set on a relatively small number of parameters as bounds on the random model generator. These parameters include the number of layers in a model, the number of faults, the range of velocity, and the dip and strike angles for each possible fault. It is believed by the authors that the randomized models produced in this manner are realistic enough to demonstrate the efficacy of neural network predictions (Figure 4)

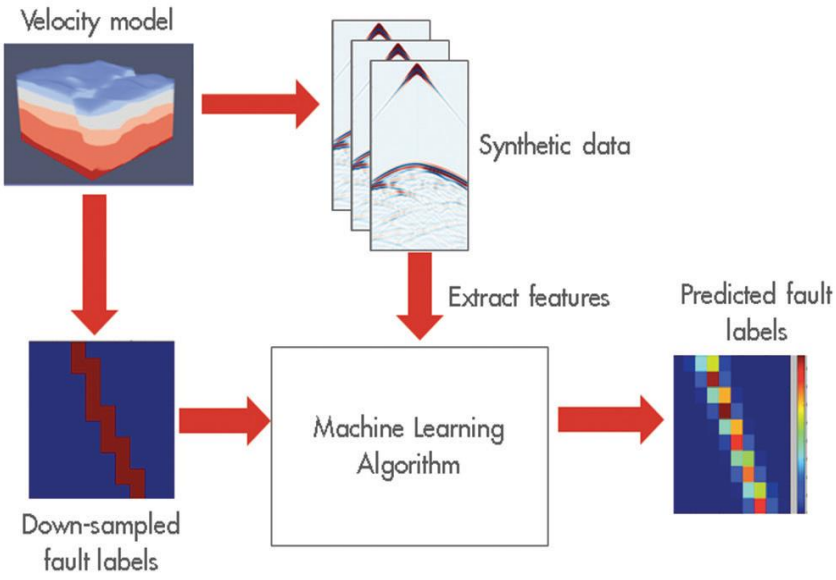


Figure 2.1.2.1: Depiction of the workflow's main tasks

Since the positions of the faults are known a priori, the labeling process is straightforward. However, complexity is added to this process when the labeling targets a subsampled grid for computational efficiency. Addition of a threshold parameter to define the labeling of the coarse grid is needed. This threshold value sets how many fine-sampled voxels inside a coarse voxel must be fault-labeled for that coarse voxel to be considered as having a fault or not. The threshold is chosen on a trial-and-error basis prior to any training. Raw seismic data is far too unrefined and redundant to be immediately useful as inputs to this neural network. The prediction performance is improved by extracting input features carefully,

taking advantage of techniques from signal processing. The amount of collected features is large and grows by orders of magnitude when more realistic models are used.

The authors have generated thousands of random velocity models with up to four faults in them, of varying strike, dip angle, and position. Their models had between three and six layers each, with velocities varying from 2000 to 4000 [m/s], with layer velocity increasing with depth. These models were 140 × 180 × 180 grid points at the sampling used for wave propagation (using the acquisition geometry described earlier) but were subsampled to 20 × 20 × 20 and 32 × 32 × 32 for labeling purposes. The raw data collected was reduced aggressively to a feature set that can fit in an NVIDIA K80 GPGPU memory.

With the generated features and labels, a variety of fully connected deep neural networks are trained. The network architecture main parameters varied from two to 20 hidden layers and 256 to 2048 units per layer. For all cases presented in Table 1, they used the Wasserstein loss function for training. The output of the networks is a subsampled 3D voxel grid, with each voxel's value indicating the likelihood of a fault being present within the voxel. Ground-truth labels on the same grid were binary valued, indicating presence or not of a fault in each voxel. The final predictions were generated by taking the likelihood values map output and applying a threshold to it, such that likelihood values above the threshold would be considered having a fault, while those below would not. As a result, a lower threshold will label more voxels as faults, while a high threshold labels less.

Here two different quantitative metrics of performance are used: intersection over union (IoU) and area under the ROC curve (AUC).

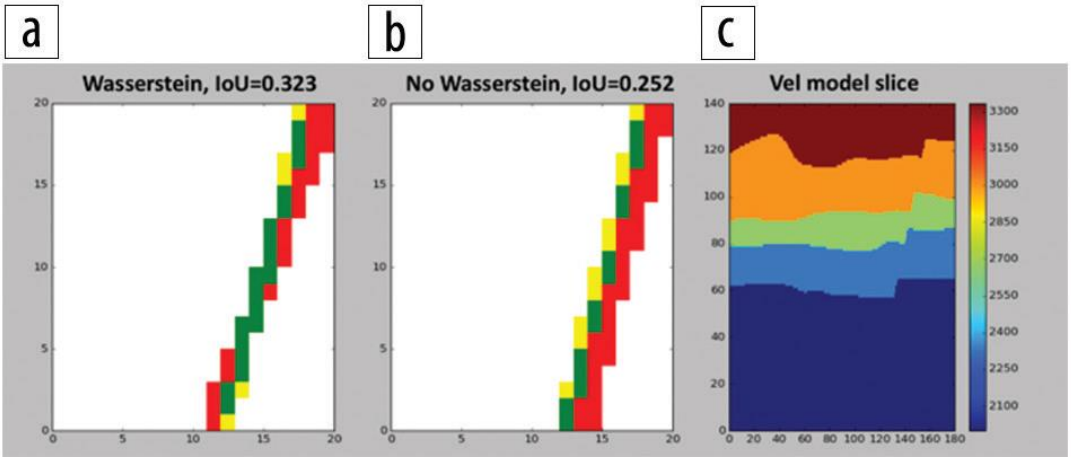


Figure 2.1.2.2: Comparison of (a) the Wasserstein- and (b) non-Wasserstein-based predictions, IoU metric (described in the article). Red areas show false positives, green shows true positives (correct predictions), and yellow shows false negative. (c) 2D slice of a 3D model. The predictions have very different IoU, where green means better.

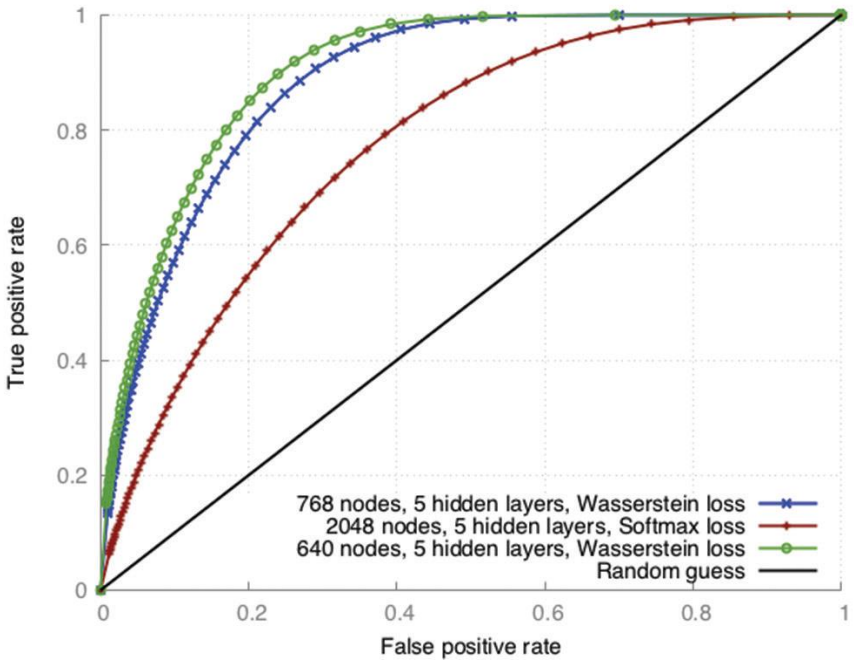


Figure 2.1.2.3: ROC curves for representative DNN architectures. The closer to the top-left corner the better.

The IoU value is a ratio of the number of voxels that are in the intersection of the ground truth and prediction, divided by the number of voxels in the union of the ground truth and prediction. This gives us an idea of how clustered or scattered a prediction is; the values range from 0 to 1, where higher values are better (Figure 5). Two predictions could theoretically have the same AUC value (Table 1) but different IoU values. The (IoU) value averaged for the entire test set of predictions, with predicted likelihoods thresholded at a value chosen to maximize the average IoU over all the predictions.

Each point of a receiver operating characteristics (ROC) curve (Figure 6) is based on the number of true positive predictions (vertical axis) and the false positive predictions (horizontal axis) for a particular threshold value. The AUC for the predictions, which describes how strong our predictor is. The value ranges from 0.5 to 1.0, where the higher the value the better.

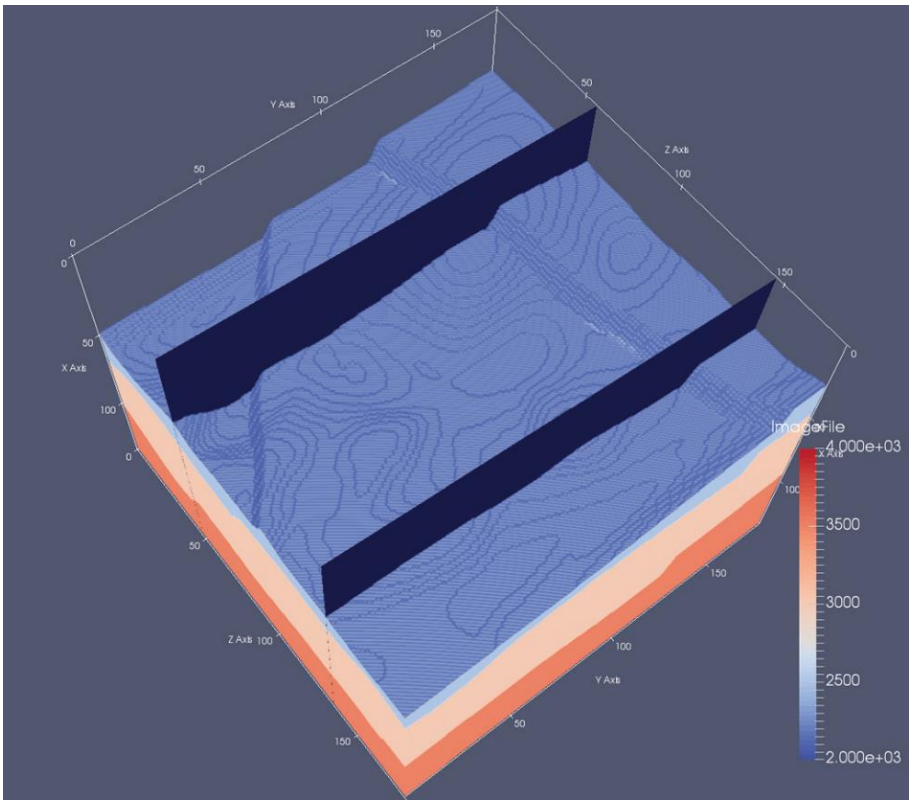


Figure 2.1.2.4: Example of 3D model with two 2D highlighted slices. In this top view, two faults can be identified.

In Table 2.1.2.1, for all data sets (one, two, and four faults in model) AUC exceeds 0.9, which approaches that needed for practical use. Also, IoU surpasses 0.3 for many experiments, which implies that the prediction can be improved in terms of spatial alignment with respect to the ground truth. The prediction grid size of experiments in Fig. 9 is 32 × 32 × 32, therefore each prediction represents a voxel of 4 × 5 × 5 in the model space.

The predictions in Figure 9 follow the expected results, plus some false positives in the bottom-right area of the back slice. However, false positives are present in the area where the faults coincide. This is in line with expectations, since a cornucopia of signals and patterns are produced in that area. This case exposes the current limit of the author's predicting resolution for complex cases. In general, the resolution is limited by the quality of the data.

To evaluate practical usability, we must address how this approach can be scaled to actual production-level seismic data sets. Current synthetic input data is based on fixed acquisition geometry. If we use this fixed geometry to train a predictor, one must bin and stack the acquired data so that it matches the acquisition used for training the model.

Table 2.1.2.1: Results obtained on several representative sets of simulated test data. The first two columns report performance metrics; the other columns describe the parameter of the experiments. All results obtained using Wasserstein loss function, with 16, 000 training models and 4000 testing models.

1024	**IoU**	**Hidden Layers**	**Nodes per Layer**	**Faults per Model**
0.902	0.311	5	768	4
0.893	0.294	5	640	4
0.836	0.220	7	640	4
0.833	0.218	8	512	4
0.854	0.246	7	512	2
0.849	0.227	6	512	2
0.820	0.219	6	512	2
0.718	0.130	4	1024	1
0.897	0.395	4	512	1
0.919	0.384	4	256	1

For this reason, we believe that using a fixed dense geometry permits us to accurately bin and stack real data to match the predictor's input parameters. Denser acquisition means more features, and as a result, more dramatic reduction of the feature space (and/or more complex neural networks) is needed.

Another area where it can improve the prediction quality is by increasing the number of voxels in our downsampled output grid such that we have better resolution in our predictions. This increase in voxel granularity means a more computationally demanding neural network and thus an increase in the cost of tuning and training.

This workflow is flexible with respect to what can be predicted. The workflow can be repurposed to build one that predicts, for instance, salt bodies instead of fault locations. It is easy to change the labeling scheme so that salt bodies are labeled before training. Since salt bodies often create strong signals (at least for the top-of-salt events) it can be expected that this geologic feature could also be identified using this method. Preliminary results along this line are promising.

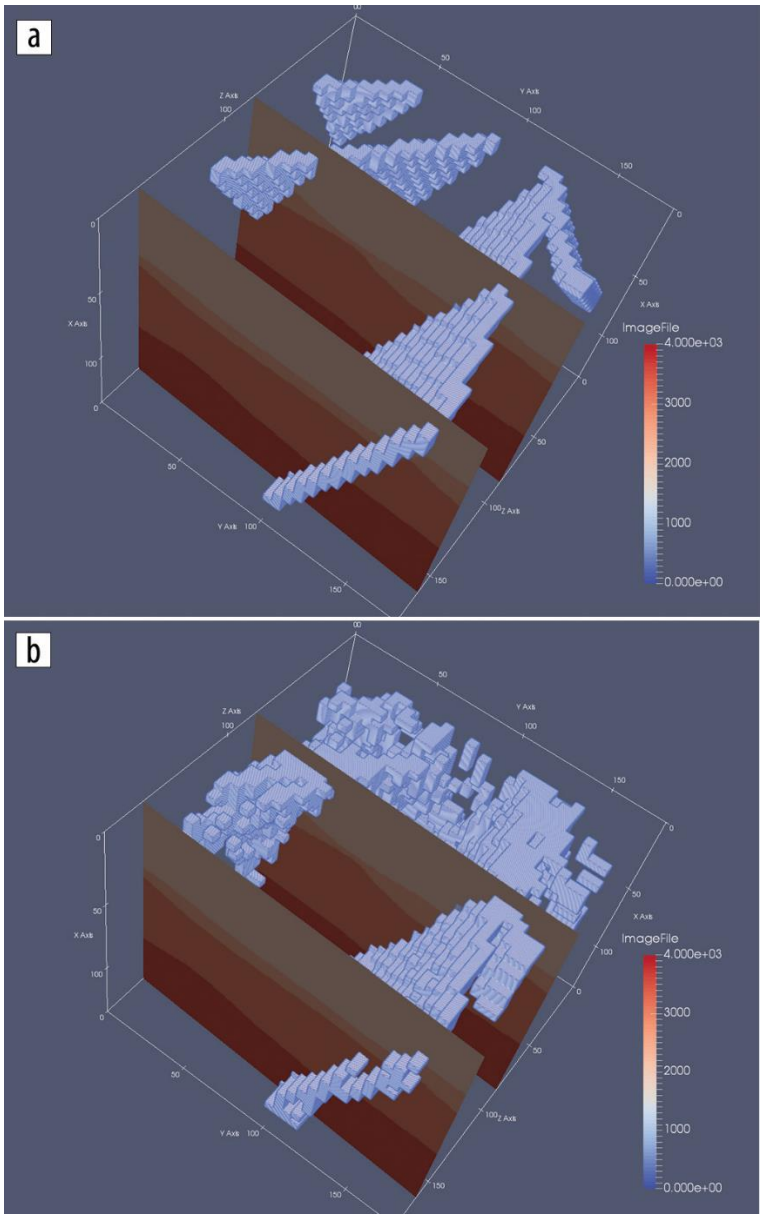


Figure 2.1.2.4: (a) Expected predictions for fault network in velocity model slices. (b) Author's DDN-based predictions.

2.1.3 Automatic salt-body classification using deep-convolutional neural network

In this paper an alternative network architecture inspired by Segnet and U-Net ,that overcomes the problem by learning to map encoder outputs to final classification labels of image dimension.The architecure used here is composed of a stack of encoder followed by a corresponding decoder stack stack which feeds into a softmax classification layer. Both the encoder and decoder are fully convolutional layers.The salt body labels are generated automatically with the aid of automatic tools.

The network is first trained on selected 2D slices,then the model is validated by predicting salt body location on other unseen slices. Here the salt body classification is treated as a semantic image segmentation problem with binary classes: the algorithm assigns a salt label to each image pixel based on the shape of the seismic image. While multiple seismic attributes can aid the salt body detection, for simplicity, we only use seismic amplitude as the input in our automated method. The network has the ability to delineate objects based on their shape despite their small size.

Compared to previous network architectures, this encoder-decoder architecture can be trained end-to-end in order to jointly optimize all the model parameters in the network. The key component of our proposed network is the decoder network which consists of a hierarchy of decoders corresponding to each encoder. The original Segnet reuses the max-pooling indices received from the encoders to perform non-linear up sampling of decoder feature maps, but this type of up sampling is prone to checkerboard artifacts (Odena et al., 2016).While these artifacts are acceptable in normal scale natural images,the issue becomes significant in seismic images which contain small scale features such as reflection signals, faults and tiny discontinuities. To overcome this issue, we adopt the resize convolution proposed by Odena et al. (2016) in our implementation. Each encoder in the encoder network performs convolution with a filter bank to produce a set of feature maps.

These are then regularized by batch normalization (Ioffe and Szegedy,2015). Element-wise rectified-linear unit ($\sigma(x) = \max\{0, x\}$) is applied as non-linear activation. Following that, we perform max-pooling and the resulting output is sub-sampled by a factor of 2. Max-pooling achieves translation invariance over small spatial shifts in the input image. The appropriate decoder in the decoder network up samples its input feature maps by resizing-interpolation. This step produces sparse feature maps which are convolved with a trainable decoder filter bank to produce dense feature maps. Batch normalizations are also applied. The high dimensional feature representation at the output of the final decoder is fed to a trainable softmax classifier. The output of the classifier is a K channel image of classification probabilities, where K is the number of classes. The k value is set to 2 where pixels are classified as inside/outside salt body.

Training process used here:

We train and test the network using SEAM Phase 1 dataset. This dataset contains a 3D seismic volume which a salt body exists in the middle. We use the 2D single-channel seismic amplitude data as our input to the network. The challenge is to identify the salt

body, from a noisy seismic image. The network can only learn from the subtle features such as high reflectivity and largely dipped boundary. We selected 8 crossline 2D slices as training data. The training labels are manual annotations generated by optimal path picking method (Wu et al., 2017). Before each epoch, the training set is shuffled. Compared to the size of the model, this is a fairly small dataset; however, the bottleneck architecture ensures that essential relationship are captured.

The model weights, or parameters, are initialized using the technique described in He et al. (2015). We use the cross-entropy loss as the objective function, and adaptive momentum descent (Adam) as optimization algorithm (Kingma and Ba, 2014) to iteratively update the model weights. After 200 epochs of training, the model achieves 98.77% global accuracy (the percentage of pixels correctly classified in the image). The figure here shows the training results of selected data samples. The salt likelihood is the probability output of the softmax classifier, and the predictions are assigned by max-likelihood class. Compared to the ground truth, the training accuracy is nearly as good as the human interpretation.

The input and output shapes are listed in the format: [samples number, image height, image width, channels number]. All samples numbers are not fixed in the network since multiple data samples can run in parallel. Note that the initial input image has 1 channel (seismic amplitude) and the final output has 2 channels (binary classification).The intermediate channels represent multiple feature maps.

Validation Tests. We first test the performance of the trained model at different crossline slices. Figure 3 shows the network output of these unseen crossline slices. Since these slices are still crossline slices, they share some similar features with the training data.

It is noticeable that some noisy artefacts appear as “holes” in the detected salt body; however, the global shape of the salt body is extracted accurately, especially on the top boundary of the salt. To visualize the result, we extract the top salt locations with > 0:6 classification probability. The right column in the figure clearly shows these top salt boundaries match seismic amplitudes accurately.

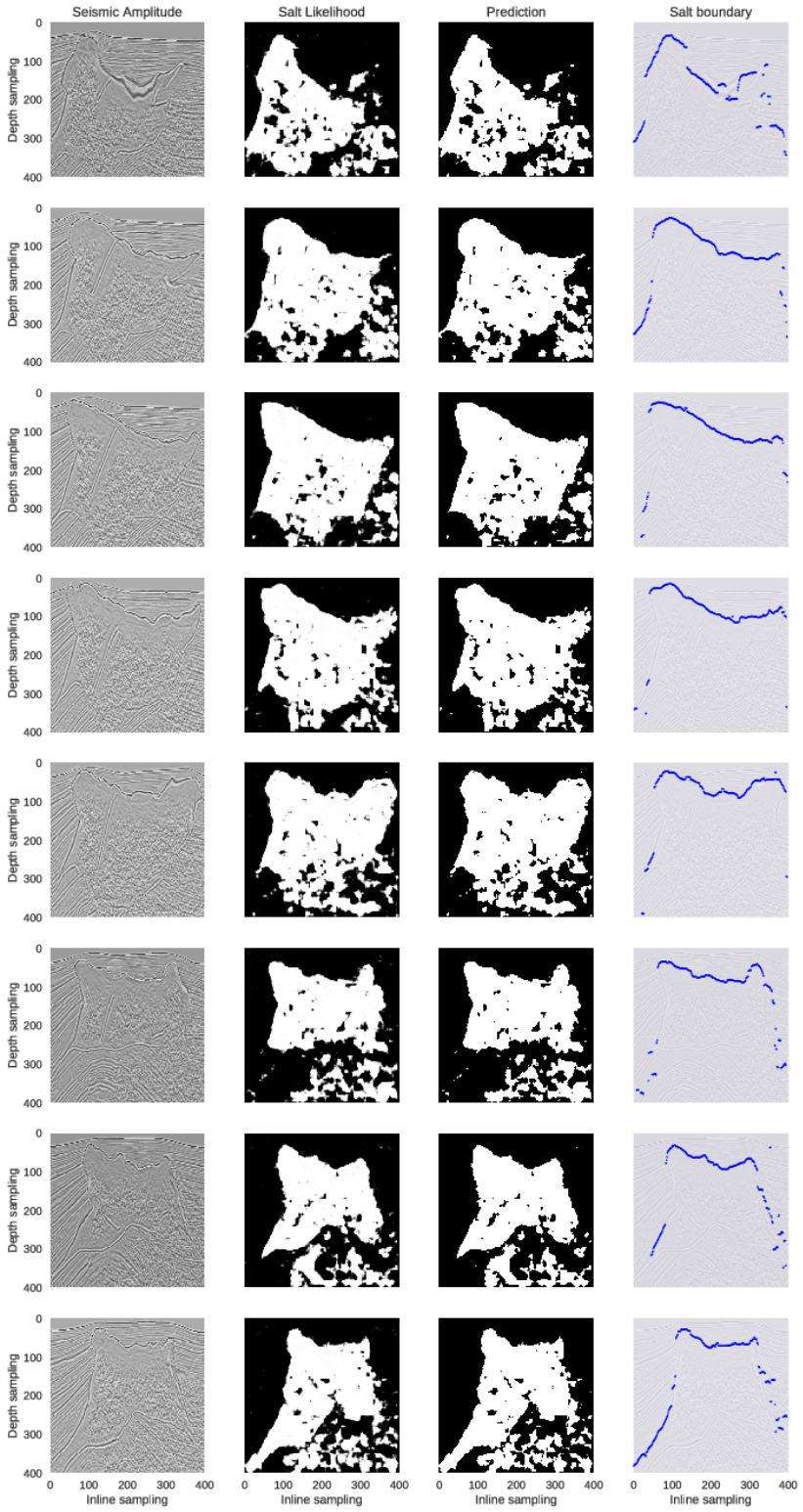


Figure 2.1.1.3.1: Selected crossline test samples and their network output visualizations.

We then test the model on inline slices. The inline slices consists of images significantly different from the training set. The performance on this test can imply whether the network successfully learn important features independently to the view perspective. Figure 4 shows the network outputs of these inline slices. The first row of Figure 4 only contains a salt

intrusion at the bottom-left corner, but the network seems to falsely classify some of the horizontal reflections as salt. The second to the seventh rows show that the salt bodies are correctly detected; however, the deeper parts of the images where seismic.

The 8 rows represent 8 crossline samples extracted at inline locations different from the ones used in training set: [25, 75, 125, 175, 225, 275, 325,375]. From left to right, the first column show seismic amplitude images; the second column show probability outputs from softmax classifier of the network; the third column show salt detection prediction results generated by max-likelihood class; the fourth column show top salt boundaries by extracting the first occurence of salt likelihood > 0:6.Images are more noisy are incorrectly assigned to salt body. The eighth row shows that although deeper image part suffers from noise, the salt body in the shallow part is still correctly delineated. Except for the first row example, all examples show good detections of the top boundary of the salt body.

Based on these results, we conclude that the proposed method can generalize well even when trained with only small dataset; the top boundary detection is the most robust prediction by the network.

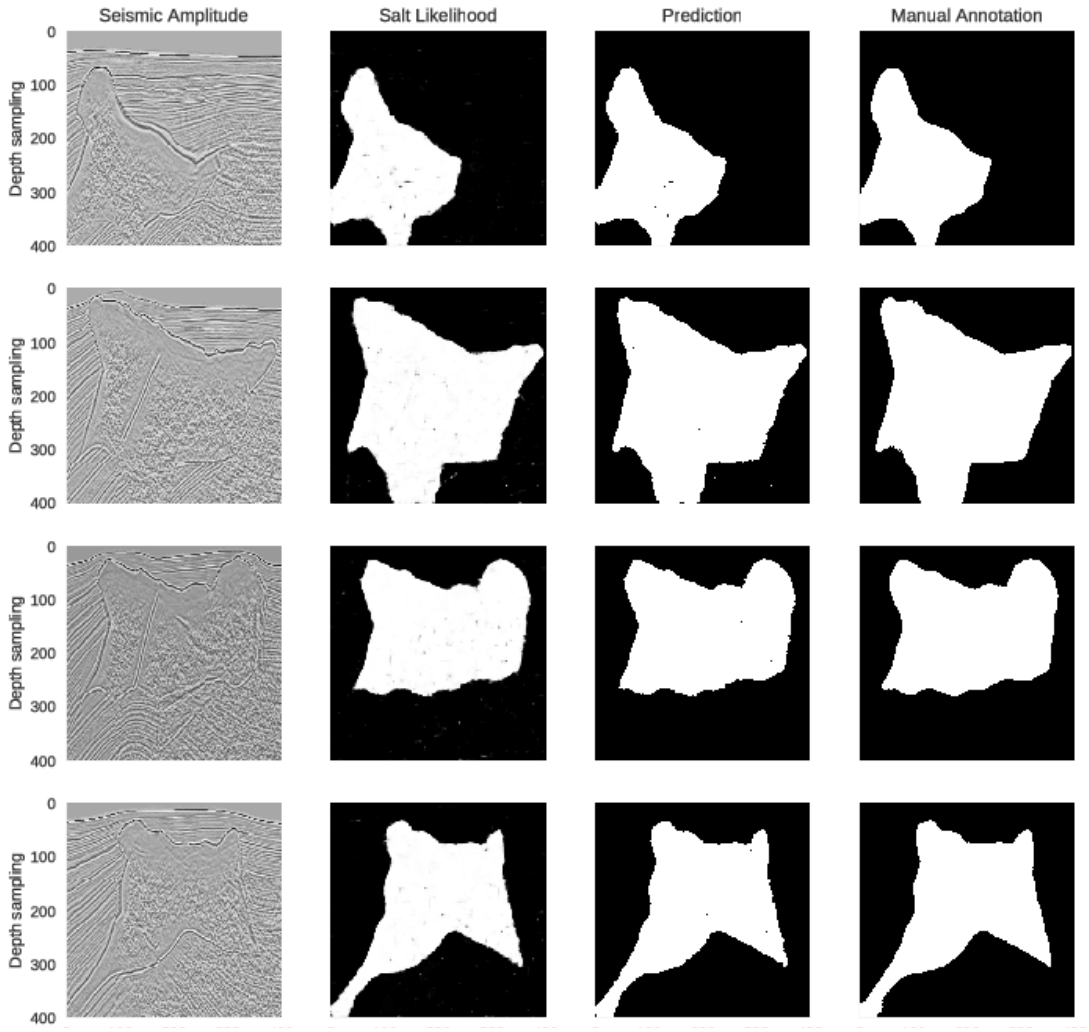


Figure 2.1.3.2 Selected training samples and their network output

2.2 Outcome of the Review – Problems Identified

2.2.1 Supervised learning to detect salt body

- The authors of [1] have used Extremely Randomized Trees, which is just a Machine Learning technique which hasn't shown state of the art results in Image Segmentation.
- The problem is that a large number of trees can make the algorithm too slow and ineffective for real-time predictions. In general, these algorithms are fast to train, but quite slow to create predictions once they are trained.
- A more accurate prediction requires more trees, which results in a slower model. In most real-world applications the random forest algorithm is fast enough, but there can certainly be situations where run-time performance is important and other approaches would be preferred.
- And of course, Random Forest is a predictive modeling tool and not a descriptive tool. That means, one will not be able to identify where the salt is located, but only that if it's present in the entire image or not.

2.2.2 Automated Fault Detection Without Seismic Processing

- To evaluate practical usability, we must address how this approach can be scaled to actual production-level seismic data sets. For instance, we have to analyze sensitivity of the predictions to acquisition geometries. Current synthetic input data is based on fixed acquisition geometry.
- Another area where we can improve the prediction quality is by increasing the number of voxels in our down sampled output grid such that we have better resolution in our predictions.
- The primary challenge going forward will be transitioning to fault networks with more complex 3D geometry. This will lead to more tuned existing networks and/or extensions to their workflows.

2.2.3 Automatic salt-body classification using deep-convolutional neural network

- Convolutional neural networks like any neural network model are computationally expensive. But, that is more of a drawback than a weakness. This can be overcome with better computing hardware such as GPUs and Neuromorphic chips.

- The weakness of CNNs lays in the amount of data you provide to them. The lesser the data, the poorer the performance.
- CNNs have millions of parameters and with small dataset, would run into an over-fitting problem because they needs massive amount of data to quench the thirst.

2.3 Proposed Work

The algorithm proposed is to use the Structure of a U-Net. U-Net is considered one of standard architectures for image classification tasks, when we need not only to segment the whole image by its class, but also to segment areas of image by class.

U-Net predicts a pixel wise segmentation map of the input image rather than classifying the input image as a whole. For each pixel in the original image, it asks the question: “To which class does this pixel belong?’’

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 Functional Requirements

Some of the functional requirements of the system are mentioned below.

- System should be able to identify pixels with salt and those without.
- Should have an acceptable accuracy.
- Should be able to classify images of any size.
- System should be able to respond within 3 seconds.

3.2 Non-Functional Requirements

Some of the non- functional requirements of the system are mentioned below.

- Portability
- Ease of Use
- Ease of Setup
- Low Maintenance Costs
- Scalable
- Availability

3.3 Software/Hardware Used

3.3.1 Hardware Requirements

Table 3.3.1.1 Hardware Requirements

Hard Disk	2 to 3 GB
Processor	Intel® Core™ i3 processor
Ram	1 GB.

3.3.2 Software Requirements

Table 3.3.2.1 Software Requirements

Operating system	Windows 7 or later, macOS, and Linux
Coding Language	Python 3.6.5
Web Technology	HTML/CSS, Javascript, Flask
Web Server	WSGI Server

Languages/Frameworks Used

Python

History of Python

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aims to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3. Due to concern about the amount of code written for Python 2, support for Python 2.7 (the last release in the 2.x series) was extended to 2020. Language developer Guido van Rossum shouldered sole responsibility for the project until July 2018 but now shares his leadership as a member of a five-person steering council.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation. A non-profit organization, the Python Software Foundation, manages Python and CPython.

Python 3.0, a major, backwards-incompatible release, was released on December 3, 2008 after a long period of testing. Many of its major features have also been backported to the backwards-compatible Python 2.6 and 2.7.

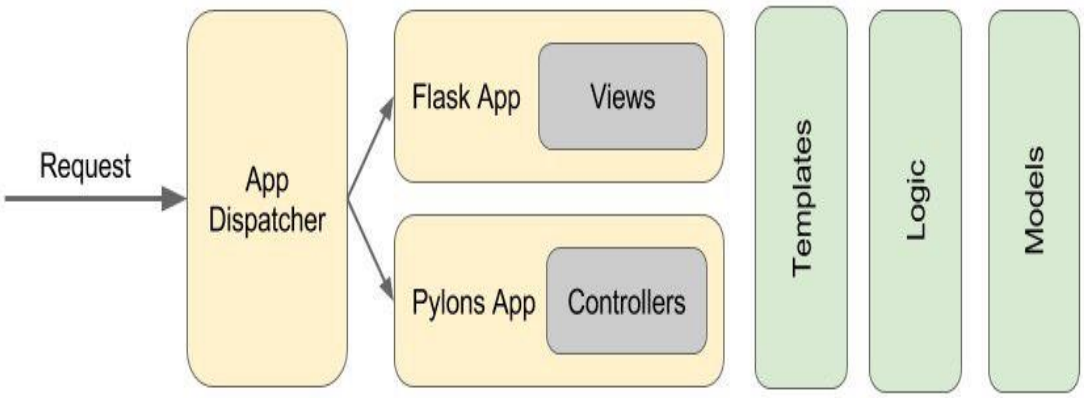
In February 1991, van Rossum published the code (labeled version 0.9.0) to alt.sources. Already present at this stage in development were classes with inheritance, exception handling, functions, and the core datatypes of list, dict, str and so on. Also in this initial release was a module system borrowed from Modula-3; Van Rossum describes the module as "one of Python's major programming units". Python's exception model also resembles Modula-3's, with the addition of an else clause.[3] In 1994 comp.lang.python, the primary discussion forum for Python, was formed, marking a milestone in the growth of Python's userbase.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Flask

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more regularly than the core Flask program.

Flask is a web application framework written in Python. Armin Ronacher, who leads an international group of Python enthusiasts named Pocco, develops it. Flask is based on Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects.



c

Keras

Keras is an open-source neural-network library, high-level neural networks API written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network model.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

eras (κέρας) means horn in Greek. It is a reference to a literary image from ancient Greek and Latin literature, first found in the Odyssey, where dream spirits (Oneiroi, singular Oneiros) are divided between those who deceive men with false visions, who arrive to Earth through a gate of ivory, and those who announce a future that will come to pass, who arrive through a gate of horn. It's a play on the words κέρας (horn) / κραίνω (fulfill), and ἐλέφας (ivory) / ἐλεφαίρομαι (deceive).

Use Keras if you need a deep learning library that:

Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).

Supports both convolutional networks and recurrent networks, as well as combinations of the two.

Runs seamlessly on CPU and GPU.

Guiding principles

User friendliness. Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

Modularity. A model is understood as a sequence or a graph of standalone, fully configurable modules that can be plugged together with as few restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions and regularization schemes are all standalone modules that you can combine to create new models.

Easy extensibility. New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.

Work with Python. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

Tensorflow

Starting in 2011, Google Brain built DistBelief as a proprietary machine learning system based on deep learning neural networks. Its use grew rapidly across diverse Alphabet companies in both research and commercial applications. Google assigned multiple computer scientists, including Jeff Dean, to simplify and refactor the codebase of DistBelief into a faster, more robust application-grade library, which became TensorFlow. In 2009, the team, led by Geoffrey Hinton, had implemented generalized backpropagation and other improvements which allowed generation of neural networks with substantially higher accuracy, for instance a 25% reduction in errors in speech recognition.

TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017.[11] While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL

extensions for general-purpose computing on graphics processing units).[12] TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors. During the Google I/O Conference in June 2016, Jeff Dean stated that 1,500 repositories on GitHub mentioned TensorFlow, of which only 5 were from Google.

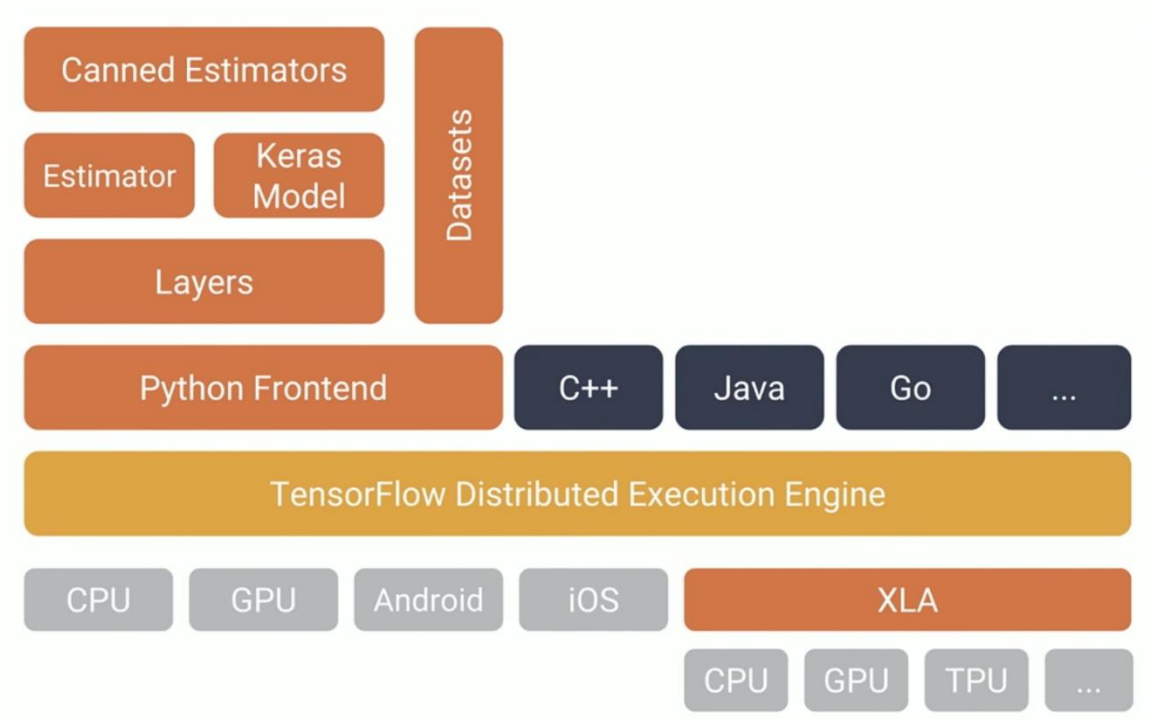


Figure 3.3.2.2: TensorFlow Model and Estimator

CHAPTER 4

SYSTEM DESIGN/METHODOLOGY

4.1 Architecture

System design is the process of defining the architecture, components, modules, interfaces and data for a system to satisfy specified requirements. System design could be seen as the application of systems theory to product development. Object-oriented analysis and design methods are becoming the most widely used methods for computer systems design. The UML has become the standard language in object-oriented analysis and design. It is widely used for modelling software systems and is increasingly used for high designing non-software systems and organizations. The look and feel of content is developed as part of graphic design; the aesthetic layout of the user interface is created as part of interface design; and the technical structure of the WebApp is modeled as part of architectural and navigational design.

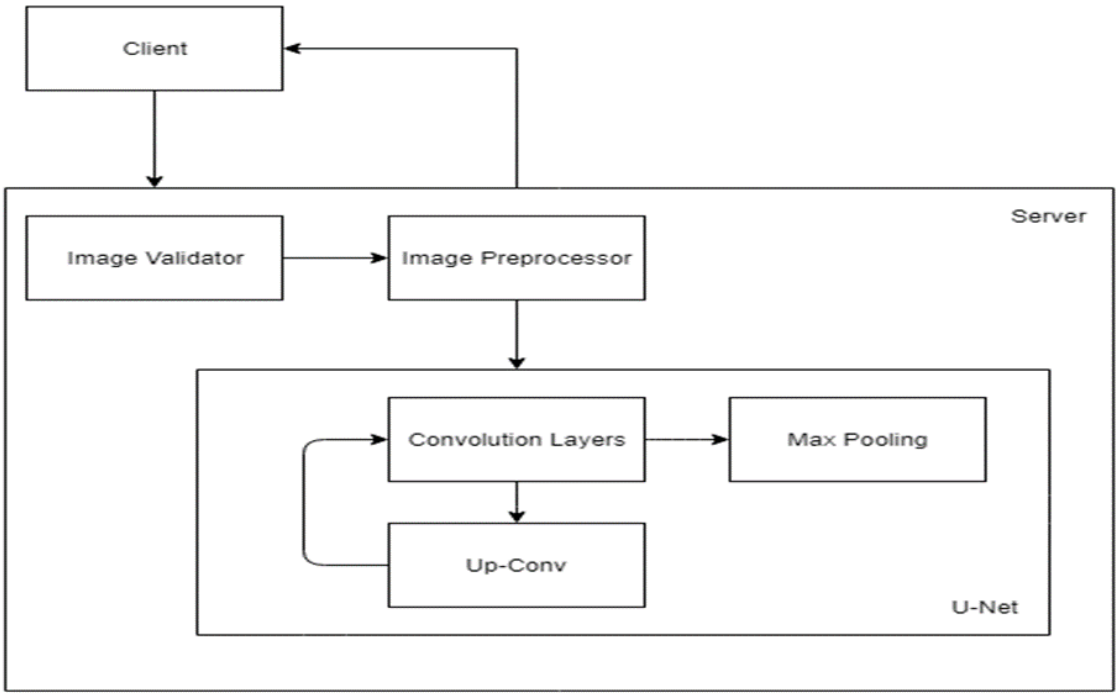


Figure 4.1.1: System Architecture

The components of the system architecture are explained below:

a. **Client:** The client here is the end user which in this case would be an organization involved in the process of hydrocarbon exploration. The client needs to upload an image of the region to be inspected and should also specify a threshold value associated with the probability of finding a salt deposit.

b. Image Preprocessor: Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them.

Image processing basically includes the following three steps.

- Importing the image with optical scanner or by digital photography.
- Analyzing and manipulating the image which includes data compression and image enhancement and spotting patterns that are not to human eyes like satellite photographs.
- Output is the last stage in which result can be altered image or report that is based on image analysis.

In the image preproccessor module the image is first converted from (X, X) into a (128, 128). If the image size of size (Y, Y) is lesser than (128, 128) then a black border of 128-Y pixels is added around the image. Further the image is also converted into a greyscale.

c. Image Validator: The role of this component is to ensure that the image uploaded by the end user is of an appropriate image format. It check for incorrect input by the user, ensuring that the model gets only images as an input.

d. Convolutional Layers: CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. The whole network has a loss function and all the tips and tricks that we developed for neural networks still apply on CNNs. Unlike neural networks, where the input is a vector, here the input is a multi-channelled image.

We take the 5*5*1 filter and slide it over the complete image and along the way take the dot product between the filter and chunks of the input image. For every dot product taken, the result is a scalar. The convolution layer is the main building block of a convolutional neural network. The convolution layer comprises of a set of independent filters. Each filter is independently convolved with the image and we end up with N feature maps of shape 25*25*1.

e. **Up-Conv**: Upsampling refers to any technique that, well, upsamples your image to a higher resolution.

   The easiest way is using resampling and interpolation. This is taking an input image, rescaling it to the desired size and then calculating the pixel values at each point using a interpolation method such as bilinear interpolation.

f. **Max Pooling**: Max pooling take the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

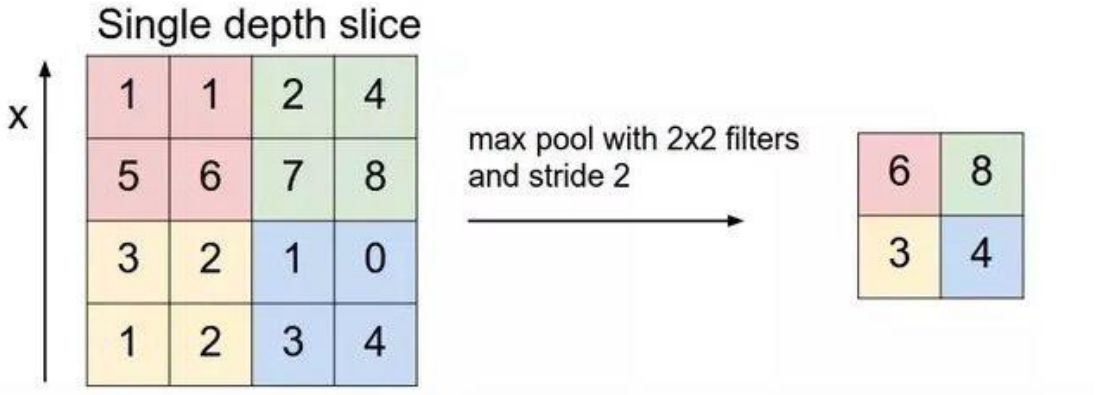


Figure 4.1.2: Max Pooling

4.2 Major Algorithm

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.

The U-Net architecture is built upon the Fully Convolutional Network and modified in a way that it yields better segmentation in medical imaging. Compared to FCN-8, the two main differences are (1) U-net is symmetric and (2) the skip connections between the downsampling path and the upsampling path apply a concatenation operator instead of a sum. These skip connections intend to provide local information to the global information while upsampling. Because of its symmetry, the network has a large number of feature

maps in the upsampling path, which allows to transfer information. By comparison, the basic FCN architecture only had number of classes feature maps in its upsampling path.

The U-Net owes its name to its symmetric shape, which is different from other FCN variants.

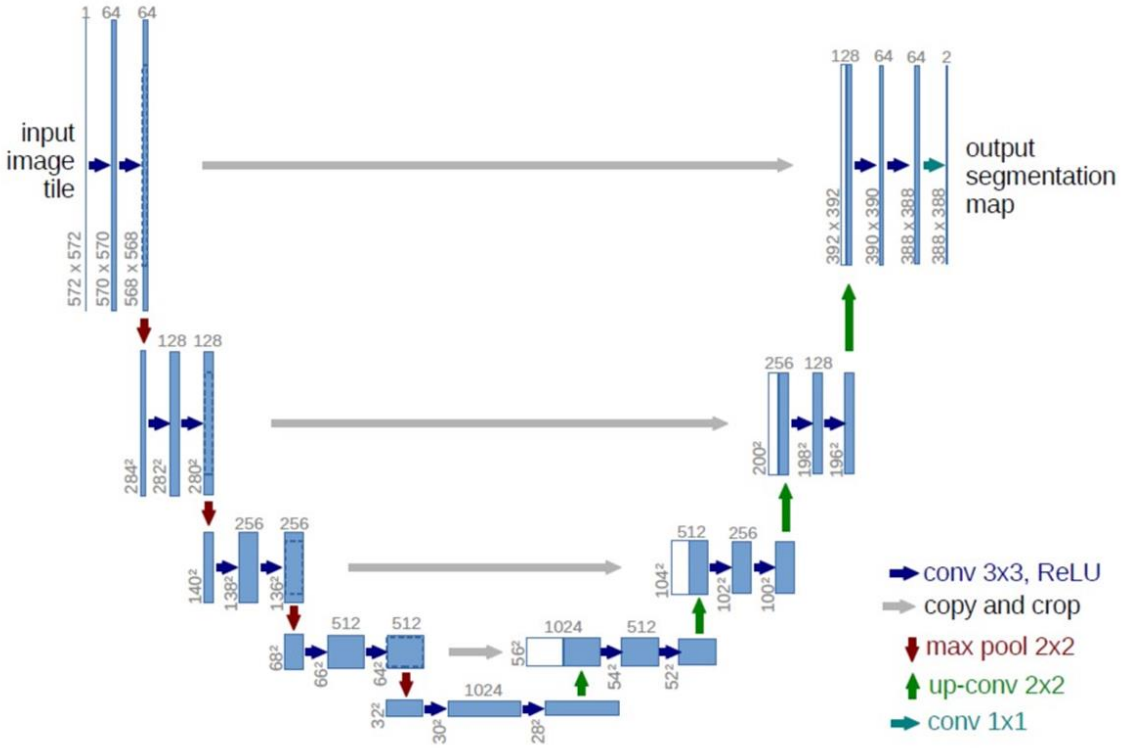


Figure 3.2.1: U-NET Architecture

U-Net architecture is separated in 3 parts:

a. **The contracting/downsampling path:** The contracting path is composed of 4 blocks. Each block is composed of

- 3x3 Convolution Layer + activation function (with batch normalization)
- 3x3 Convolution Layer + activation function (with batch normalization)
- 2x2 Max Pooling

Note that the number of feature maps doubles at each pooling, starting with 64 feature maps for the first block, 128 for the second, and so on. The purpose of this contracting path is to capture the context of the input image in order to be able to do segmentation. This coarse contextual information will then be transfered to the upsampling path by means of skip connections.

b. Bottleneck: This part of the network is between the contracting and expanding paths. The bottleneck is built from simply 2 convolutional layers (with batch normalization), with dropout.

c. **The expanding/upsampling path:** The expanding path is also composed of 4 blocks. Each of these blocks is composed of

- Deconvolution layer with stride 2
- Concatenation with the corresponding cropped feature map from the contracting path
- 3x3 Convolution layer + activation function (with batch normalization)
- 3x3 Convolution layer + activation function (with batch normalization)

The purpose of this expanding path is to enable precise localization combined with contextual information from the contracting path.

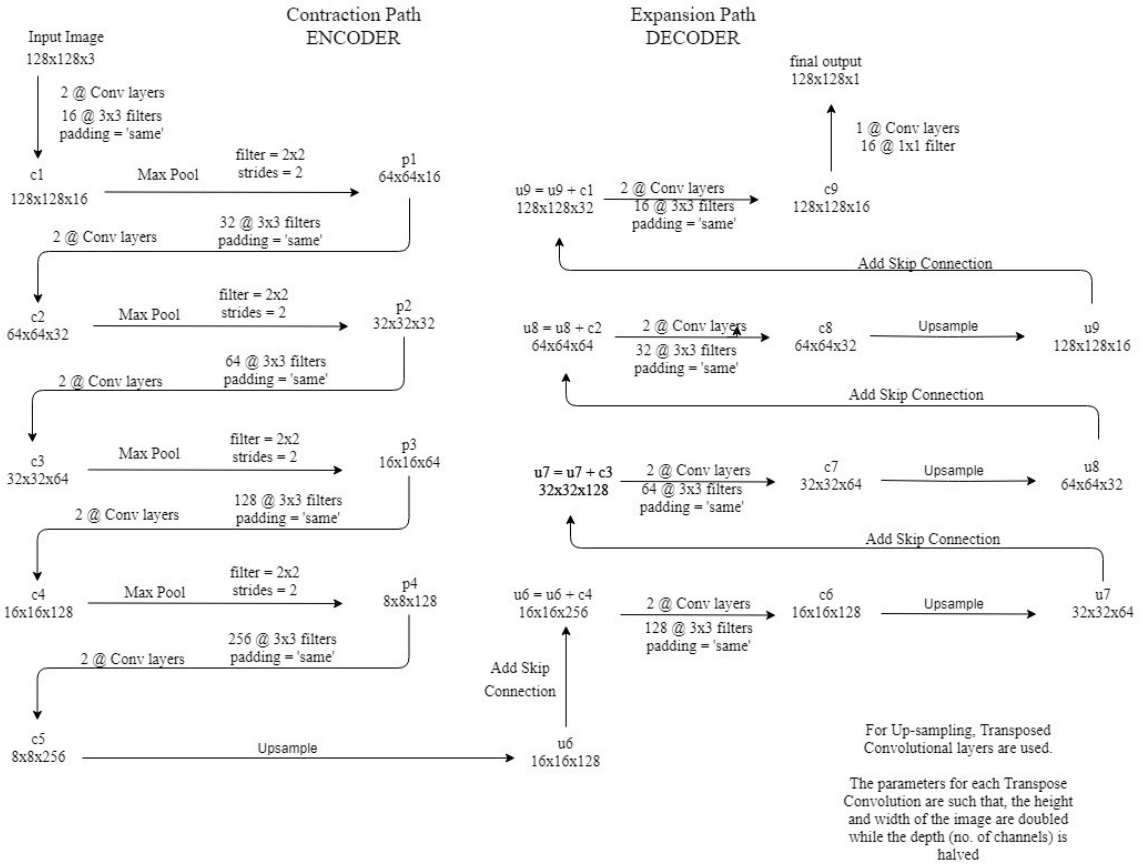


Figure 4.2.2: Detailed U-NET Architecture

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.

- 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)

CHAPTER 5

IMPLEMENTATION

5.1 Module 1: Image Preprocessor

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them.

Image processing basically includes the following three steps.

- Importing the image with optical scanner or by digital photography.
- Analyzing and manipulating the image which includes data compression and image enhancement and spotting patterns that are not to human eyes like satellite photographs.
- Output is the last stage in which result can be altered image or report that is based on image analysis.

In the image preproccessor module the image is first converted from (X, X) into a (128, 128). If the image size of size (Y, Y) is lesser than (128, 128) then a black border of 128-Y pixels is added around the image. Further the image is also converted into a greyscale.

5.2 Module 2: Image Validator

The role of this component is to ensure that the image uploaded by the end user is of an appropriate image format. It check for incorrect input by the user, ensuring that the model gets only images as an input.

5.3 Module 3: Convolutional Layers

CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. The whole network has a loss function and all the tips and tricks that we developed for neural networks still apply on

CNNs. Unlike neural networks, where the input is a vector, here the input is a multi-channelled image.

There are two inputs to a convolutional operation:

- A 3D volume (input image) of size (nin x nin x channels)
- A set of 'k' filters (also called as kernels or feature extractors) each one of size (f x f x channels), where f is typically 3 or 5.

The output of a convolutional operation is also a 3D volume (also called as output image or feature map) of size (nout x nout x k).
The relationship between nin and nout is as follows:

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

Convolution operation can be visualized as follows:

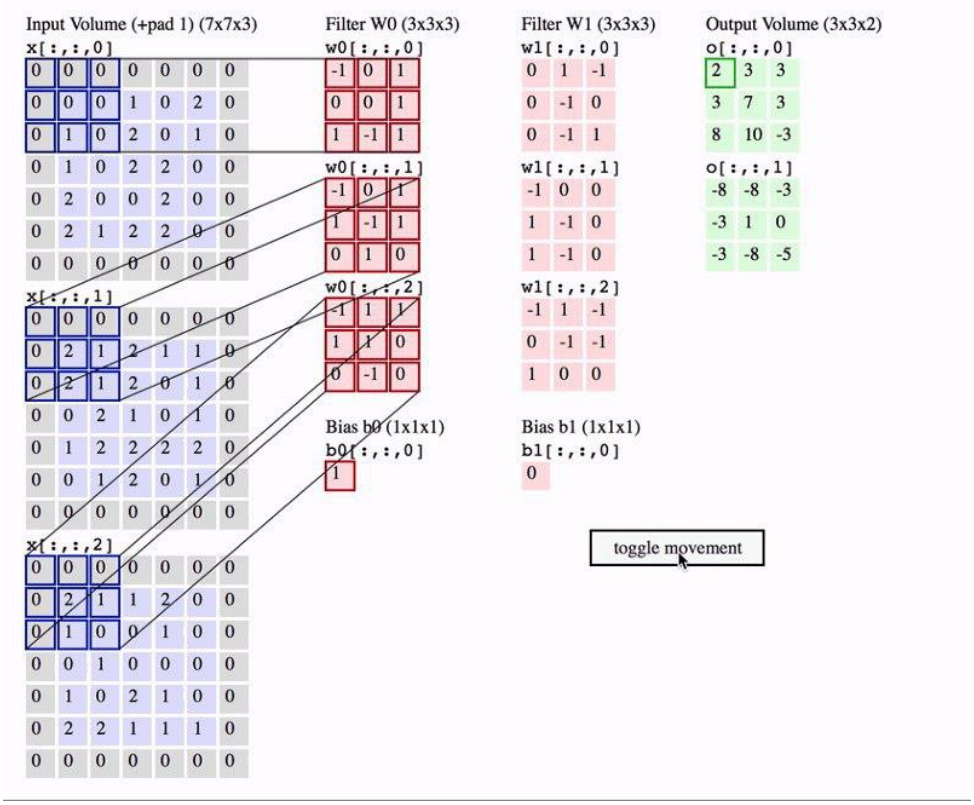


Figure 4.3.1: Convolution Operation

In the above image, we have an input volume of size 7x7x3. Two filters each of size 3x3x3. Padding =0 and Strides = 2. Hence the output volume is 3x3x2. If you are not comfortable with this arithmetic then you need to first revise the concepts of Convolutional Networks before you continue further.

One important term used frequently is called as the Receptive filed. This is nothing but the region in the input volume that a particular feature extractor (filter) is looking at. In the above GIF, the 3x3 blue region in the input volume that the filter covers at any given instance is the receptive field. This is also sometimes called as the context.

To put in very simple terms, receptive field (context) is the area of the input image that the filter covers at any given point of time.

5.4 Module 4: Up-Conv

Upsampling refers to any technique that, well, up samples your image to a higher resolution.

The easiest way is using resampling and interpolation. This is taking an input image, rescaling it to the desired size and then calculating the pixel values at each point using a interpolation method such as bilinear interpolation.

As stated previously, the output of semantic segmentation is not just a class label or some bounding box parameters. In-fact the output is a complete high resolution image in which all the pixels are classified.

Thus if we use a regular convolutional network with pooling layers and dense layers, we will lose the “WHERE” information and only retain the “WHAT” information which is not what we want. In case of segmentation we need both “WHAT” as well as “WHERE” information.

Hence there is a need to up sample the image, i.e. convert a low resolution image to a high resolution image to recover the “WHERE” information.

In the literature, there are many techniques to up sample an image. Some of them are bi-linear interpolation, cubic interpolation, nearest neighbor interpolation, unpooling, transposed convolution, etc. However in most state of the art networks, transposed convolution is the preferred choice for up sampling an image.

5.5 Module 5: Max Pooling

Max pooling take the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

In simple words, the function of pooling is to reduce the size of the feature map so that we have fewer parameters in the network.

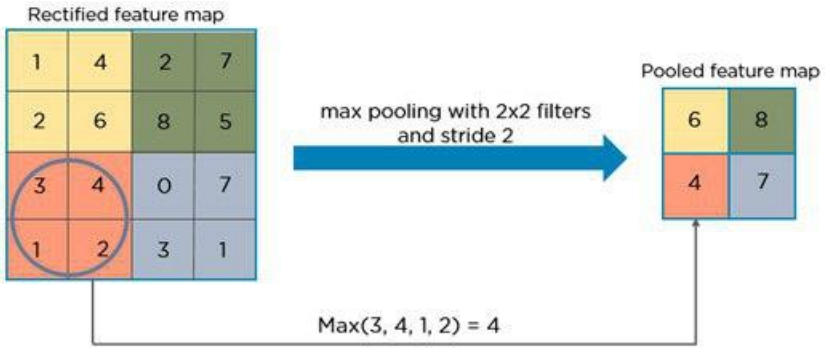


Figure 5.5.1: Max Pooling

Basically from every 2x2 block of the input feature map, we select the maximum pixel value and thus obtain a pooled feature map. Note that the size of the filter and strides are two important hyper-parameters in the max pooling operation.

The idea is to retain only the important features (max valued pixels) from each region and throw away the information which is not important. By important, I mean that information which best describes the context of the image.

A very important point to note here is that both convolution operation and specially the pooling operation reduce the size of the image. This is called as down sampling. In the above example, the size of the image before pooling is 4x4 and after pooling is 2x2. In fact down sampling basically means converting a high resolution image to a low resolution image.

Thus before pooling, the information which was present in a 4x4 image, after pooling, (almost) the same information is now present in a 2x2 image.

Now when we apply the convolution operation again, the filters in the next layer will be able to see larger context, i.e. as we go deeper into the network, the size of the image reduces however the receptive field increases.

For example, below is the LeNet 5 architecture:

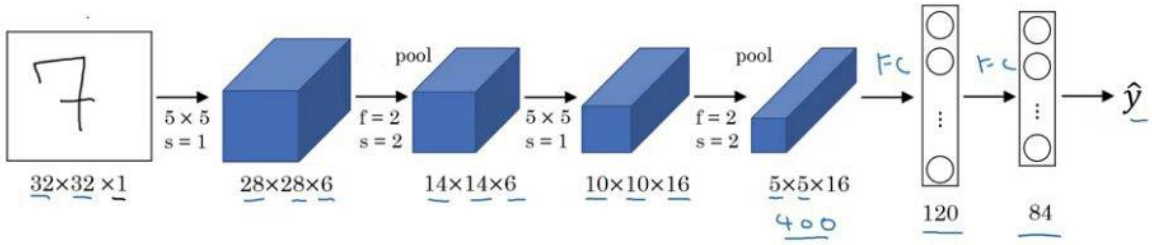


Figure 5.5.2: LeNet Architecture

Notice that in a typical convolutional network, the height and width of the image gradually reduces (down sampling, because of pooling) which helps the filters in the deeper layers to focus on a larger receptive field (context). However the number of channels/depth (number of filters used) gradually increase which helps to extract more complex features from the image.

Intuitively we can make the following conclusion of the pooling operation. By down sampling, the model better understands "WHAT" is present in the image, but it loses the information of "WHERE" it is present.

5.6 Module 6: Transposed Convolution

Transposed convolution (sometimes also called as deconvolution or fractionally strided convolution) is a technique to perform up sampling of an image with learnable parameters.

I will not describe how transpose convolution works because Naoki Shibuya has already done a brilliant job in his blog Up sampling with Transposed Convolution. I strongly recommend you to go through this blog (multiple times if required) to understand the process of Transposed Convolution.

However, on a high level, transposed convolution is exactly the opposite process of a normal convolution i.e., the input volume is a low resolution image and the output volume is a high resolution image.

In the blog it is nicely explained how a normal convolution can be expressed as a matrix multiplication of input image and filter to produce the output image. By just taking the transpose of the filter matrix, we can reverse the convolution process, hence the name transposed convolution.

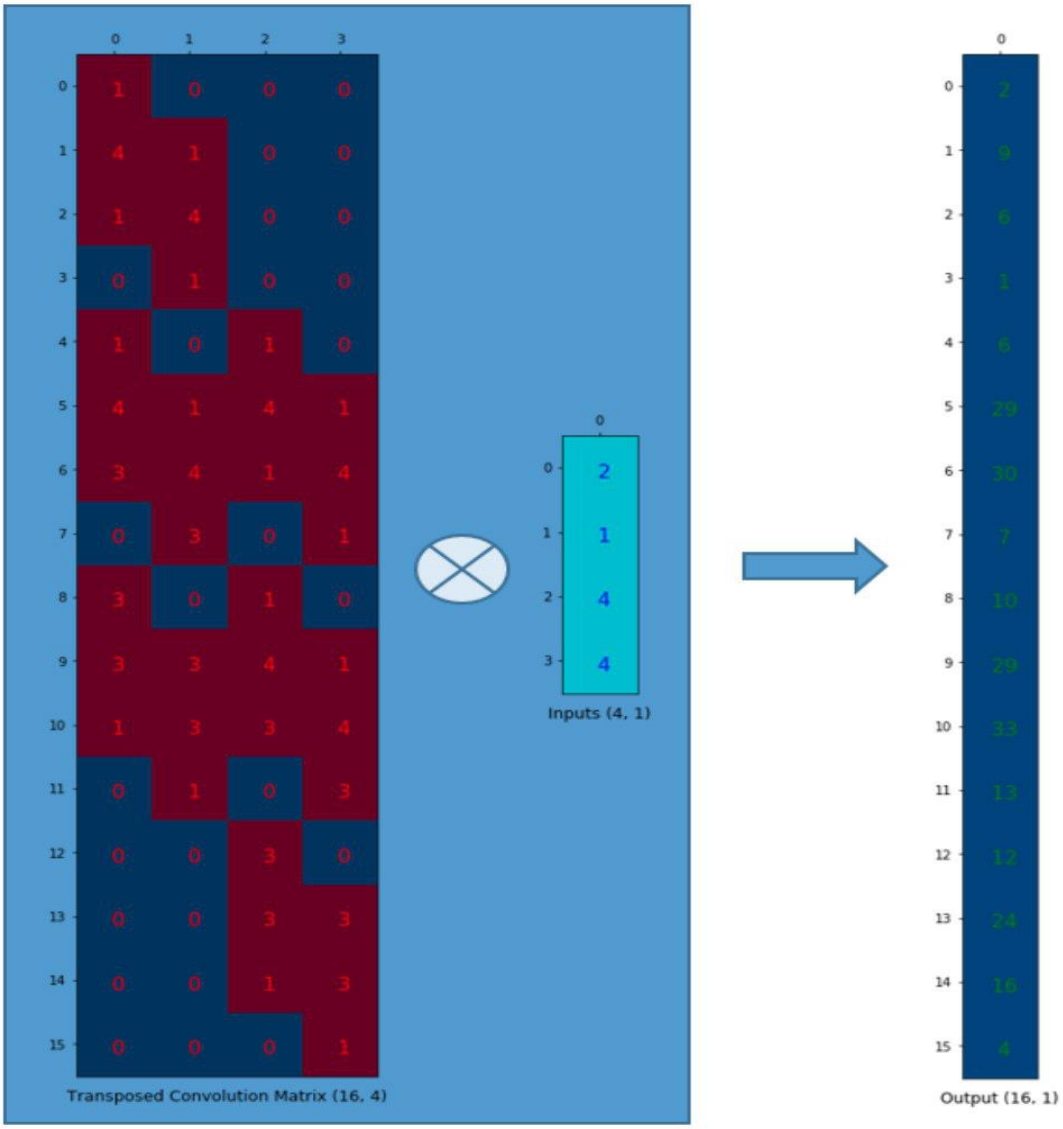


Figure 5.6.1: Transpose Convolution

- Convolution and pooling operations down sample the image, i.e. convert a high resolution image to a low resolution image
- Max Pooling operation helps to understand “WHAT” is there in the image by increasing the receptive field. However it tends to lose the information of “WHERE” the objects are.
- In semantic segmentation it is not just important to know “WHAT” is present in the image but it is equally important to know “WHERE” it is present. Hence we need a way to up sample the image from low resolution to high resolution which will help us restore the “WHERE” information.
- Transposed Convolution is the most preferred choice to perform up sampling, which basically learns parameters through back propagation to convert a low resolution image to a high resolution image.

Data Flow Model

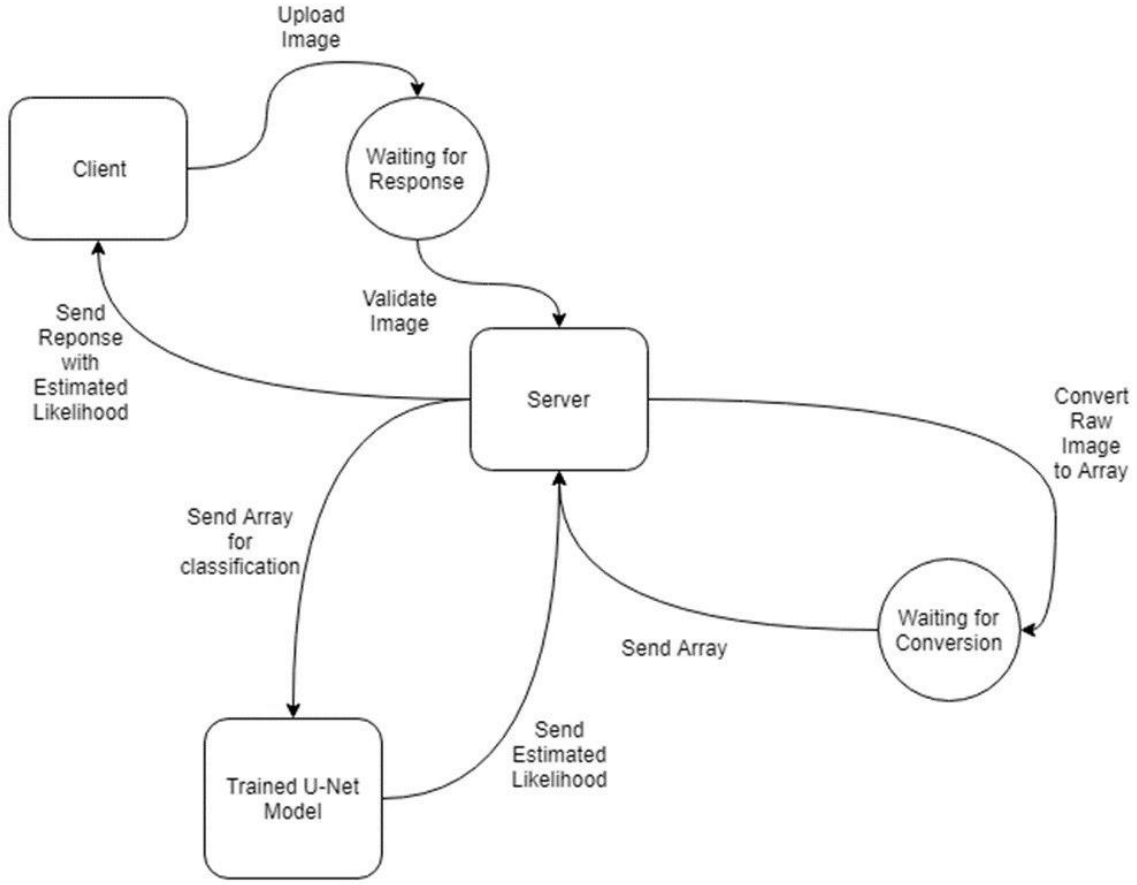


Figure 5.6.2 Data Flow Diagram

Step 1: The client accesses the website where one can securely upload a seismic image using the POST request method.

Step 2: In some cases, the client may choose to enter a particular threshold value.

Step 3: The user should click on the predict button, in order to send the data entered by him/her to the server for prediction.

Step 4: The server validates whether the data entered by the user is an image or not. If it is not an image it just closes the connection.

Step 5: If the data is validated successfully, the image is converted into the .png format.

Step 6: Once the .png image has been generated, the server upscales/downscales the image to 128x128 pixels depending upon the requirement.

Step 7: The image is converted into an array of pixel values, whose values range from 0 to 1.

Step 8: The array is sent as an input to the trained model.

Step 9: The trained model returns a feature map which contains the probability of each pixel having salt.

Step 10: A heat map is generated using this feature map and is sent to the user along with the proportion of salt.

Sequence Diagram

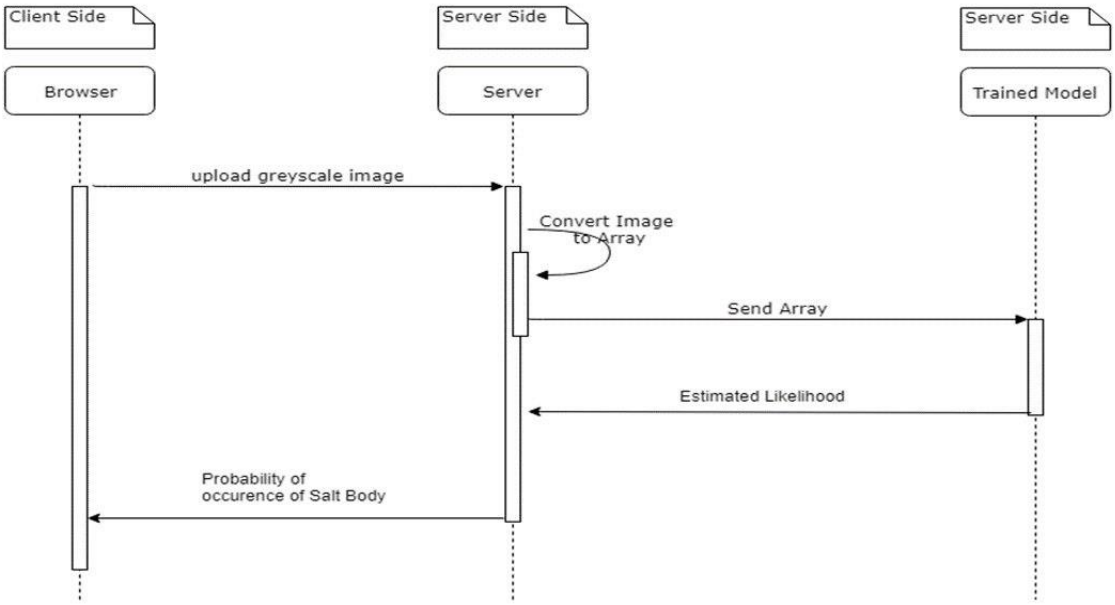


Figure 5.6.3 Sequence Diagram

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios. A sequence diagram shows, as parallel vertical lines, different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner. Figure 8 shows the sequence diagram which depicts the following steps:

1. Upload Greyscale Image.

2. Convert Image to Array.

3. Send Array to the trained model.

4. Return Feature map along with an estimated likelihood.

5. Return the results back to the client.

Use Case Diagram

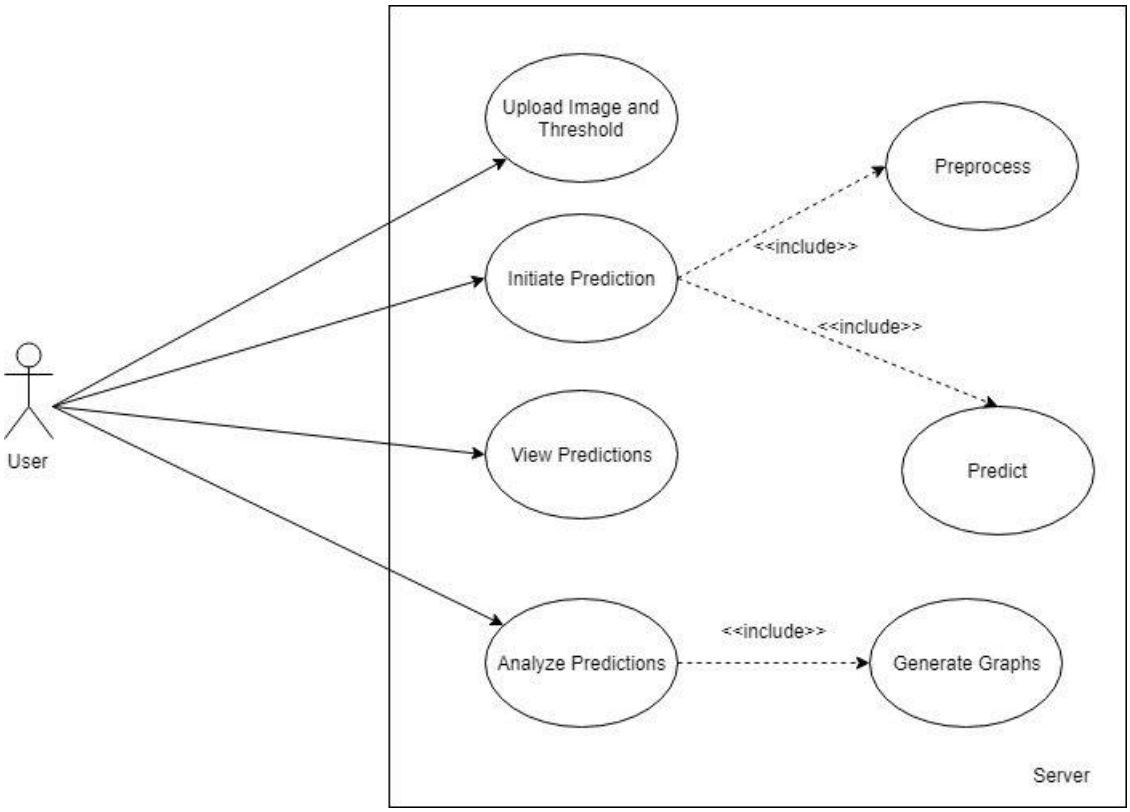


Figure 5.6.4 Use Case Diagram

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system

5.7 User Defined Functions

A User- Defined Function (UDF) is a function provided by the user of a program or environment, in a context where the usual assumption is that functions are built into the program or environment. Below mentioned are the user- defined functions used.

Table 5.7.1: List of UDFs

Function Name	**Input Parameters**	**Return type**	**Description**
get_image()	Image(X,X)	Boolean	Used to upload the image. Image is also converted to png file, then converted to (101,101) pixels. Stored in uploads folder. File name of the image is added to the session variable.
get_thres()	String	Boolean	Takes Threshold value and adds the integer value to session variable.
pre_process_me()	String	List[][]	Converts Image to Array using Keras. Resizes the array to 128,128. Calculate cumulative mean, cumulative sum of all values across X. Subtract cumulative mean from cumulative sum. Divide cumulative sum by the standard deviation. Divide each value in X by 255.
predict_me()	List[][],Integer	List[][]	Send input to the trained model and predict. Convert the prediction based on the threshold entered. Find salt proportion in given region.
get_mask_graph()	List[]	JSON serialized object	It return the final JSON graph which contain the data about the salt proportions in the input image.
salt_proportion()	List[][], Integer	Float	If element in the Y is greater than threshold then it is replaced with 255 else 0. Then calculates the number of 255's in the 2D list and divides by 15744.
gen_res()	List[],List[],List[],List[]	Resulting PNG file.	For each image name in the list of images, the image and ground truth mask is retrieved. Each image is also preprocessed using the **pre_process_me()** and its output is sent to the model using **predict_me().** For each different threshold a new output is predicted.

scatter_random()	List[], List[]	JSON serialized object.	Plots a scatter plot between depth and salt proportion. The X axis is the proportion of salt and Y axis is the depth.

Chapter 6

Results and Discussion

6.1 Testing

6.1.1 Unit Testing

Table 6.1.1.1: Unit Testing

Sl.No.	Test Case	Input	Expected Output	Actual Results	Remarks
1	All set of fields in the upload form are filled by the user and submitted.	Seismic Image, Threshold value.	Upload should be successful	Upload Successful.	Pass
2	Image Validation-Accepting only images as input.	Seismic Image of any size.	Upload should be successful.	Upload Successful.	Pass
3	Image Validation-Accepting only image as input.	Word document was uploaded as input.	Upload should be unsuccessful.	Upload Unsuccessful.	Pass
4	Image Preprocessing-Accept Image of any size and convert it to 101 x 101.	Seismic Image of any size.	Image of size 101 x 101.	Image of size 101 x 101.	Pass
5	Adding a border of 14 pixels around the entire image.	Seismic image of size 101 x 101.	An Image of size 128 x 128 with 14 pixels as border.	Image of size 128 x 128 that includes the border.	Pass
6	Conversion of Image to an Array containing values in the interval [0,1].	Pre-processed image of size 128 x 128.	A 3D Matrix of size 128 x 128 x 2.	A 3D Matrix of size 128 x 128 x 2.	Pass

6.1.2 Integration Testing

Table 6.1.1.2: Integration Testing

Sl.No.	Test Case	Input	Expected Output	Actual Results	Remarks
1	Image Uploading and Image Validation	Seismic Image of any size.	Upload should be successful.	Upload Successful.	Pass
2	Image Pre-processing and adding border	Seismic Image of any size.	An Image of size 128 x 128 with 14 pixels as border.	Image of size 128 x 128 that includes the border.	Pass
3	Convert to Array and send as input to model	Pre-processed image of size 128 x 128.	A 2D array of size 128 x 128.	A 2D array of size 128 x 128.	Pass

6.1.3 System Testing

Table 6.1.1.3: System Testing

Sl.No.	Test Case	Input	Expected Output	Actual Results	Remarks
1	System should be able to identify pixels with salt and those without.	Seismic Image, Threshold value.	Identified pixels should be comparable with the ground truth.	Identified pixels are comparable with the ground truth.	Pass
2	Should be able to classify images of any size.	Seismic Image of any size.	The salt mask should be shown on the website.	The salt mask is displayed on the website.	Pass
4	System should be able to respond within 3 seconds.	Seismic Image, Threshold value.	Average Service Time should be	Average Service Time is	Pass

			below 3 seconds.	2.774 seconds.	

6.2 Results

6.2.1 Screenshots

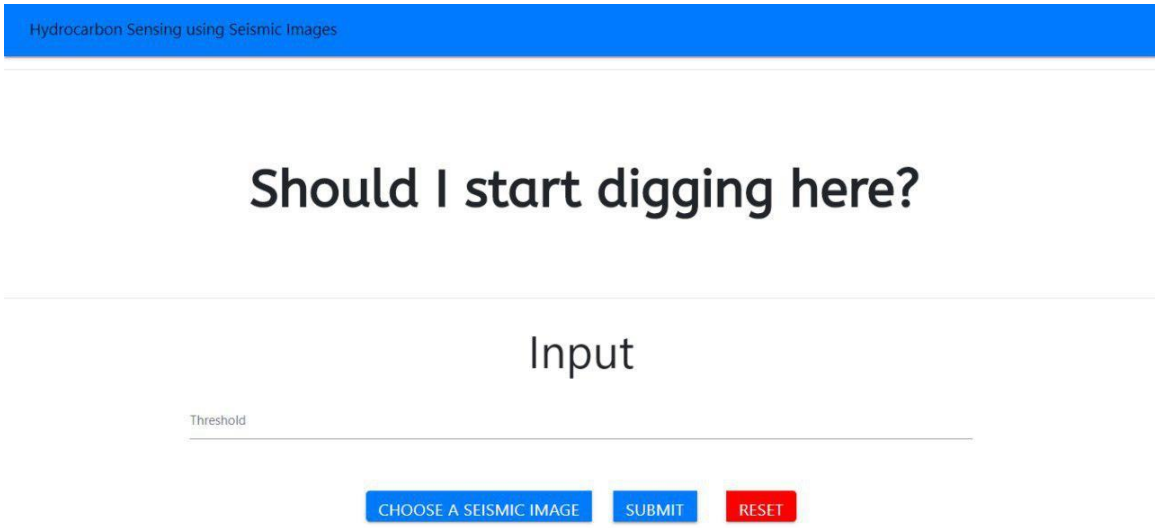


Figure 6.2.1.1: Home page: Upload Form

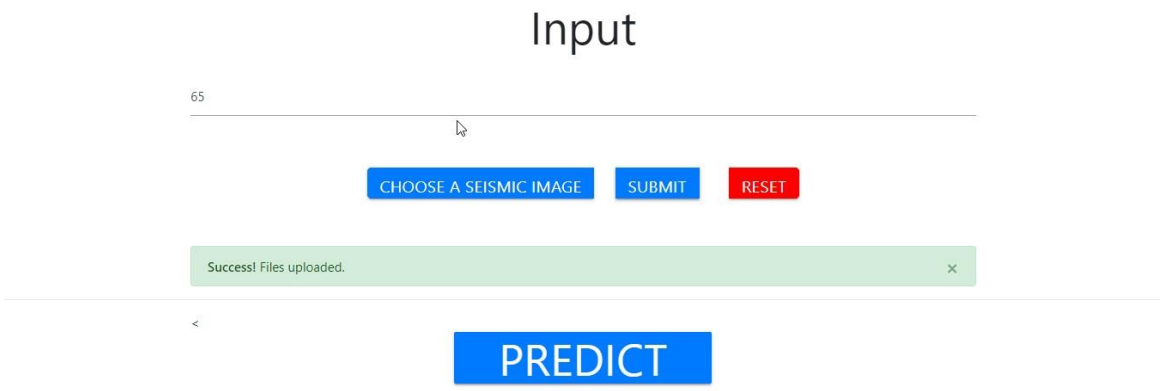


Figure 6.2.1.2: File uploaded and Threshold value entered

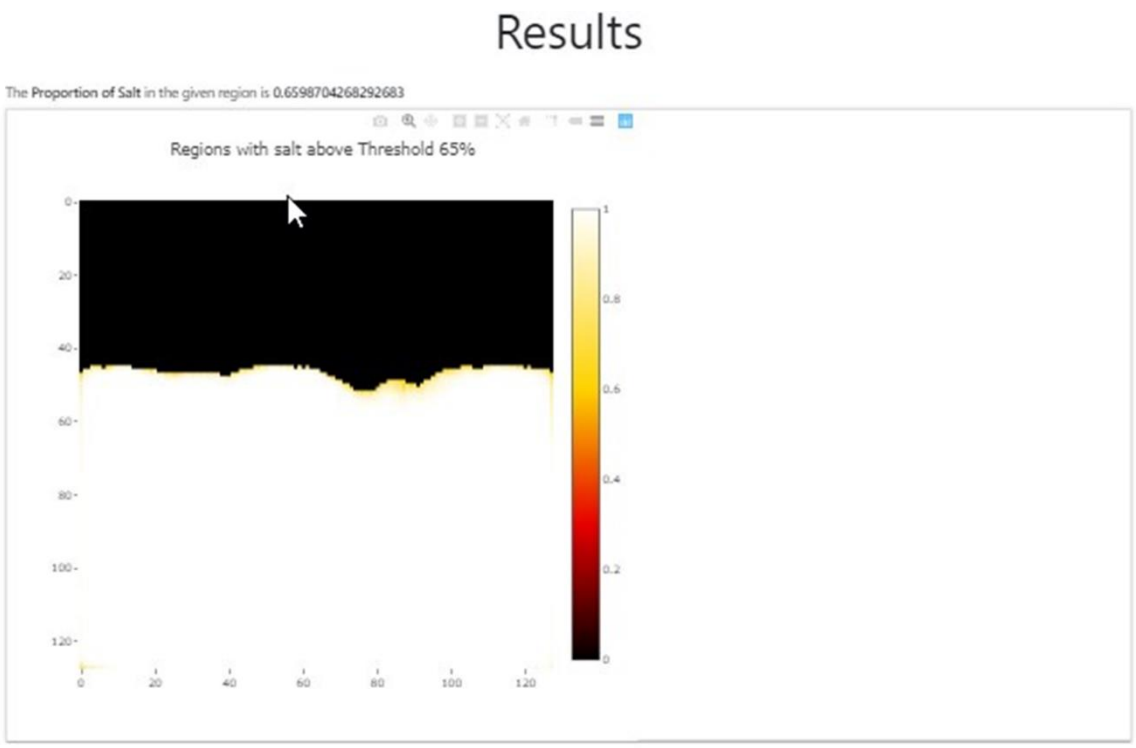


Figure 6.2.1.3: Results Section

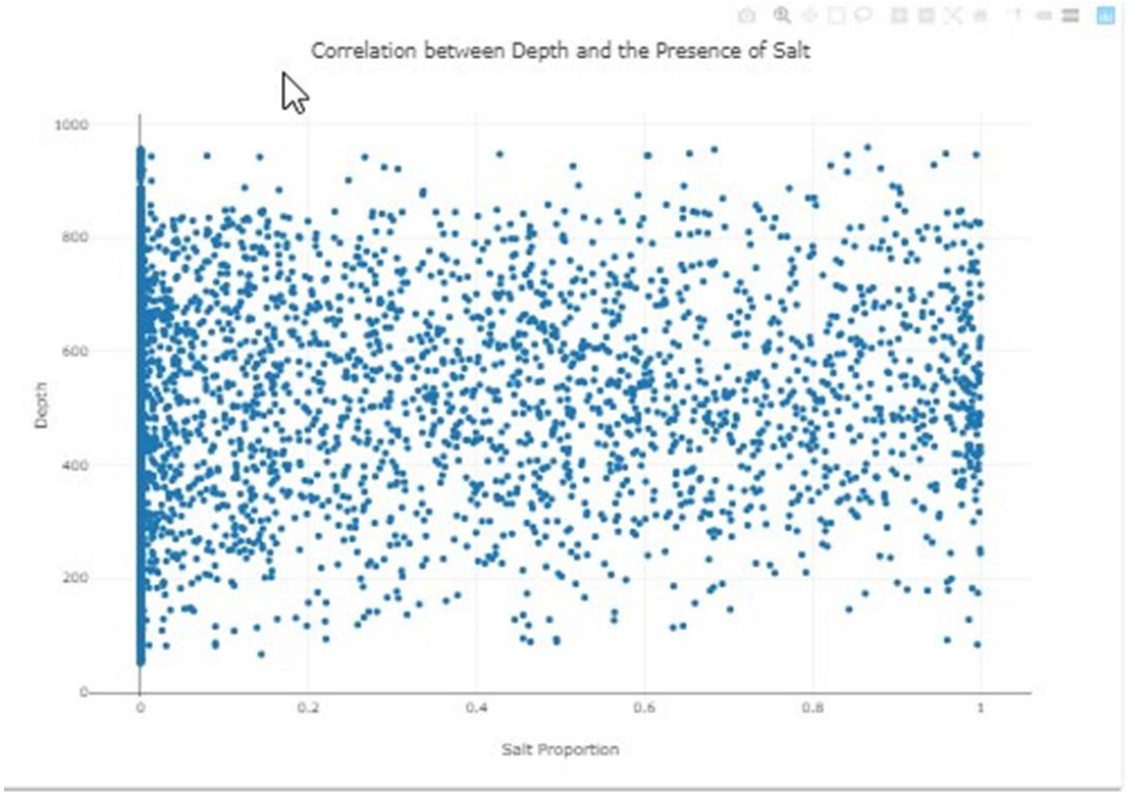


Figure 6.2.1.4: Scatter Plot under Results Section

6.2.2 Inferences from the results

Figure 6.2.1 shows the results obtained by the usage of U-Nets for salt classification. The first column consists of the raw seismic images the model takes as input. The second

column shows the mask image which is the ground truth that is used as a comparison for our model's prediction. Both of the above-mentioned columns show the X-axis and Y-axis ranging from 0 to 101. This is due to the fact that our dataset consists of images of the same size. The next three columns show the predicted mask image by the U-Net model. We have used three different threshold values 30%, 50% and 90%. It can be observed that these threshold values do make a difference in the prediction. The threshold value is directly proportional the probability of finding salt in the non-black regions.

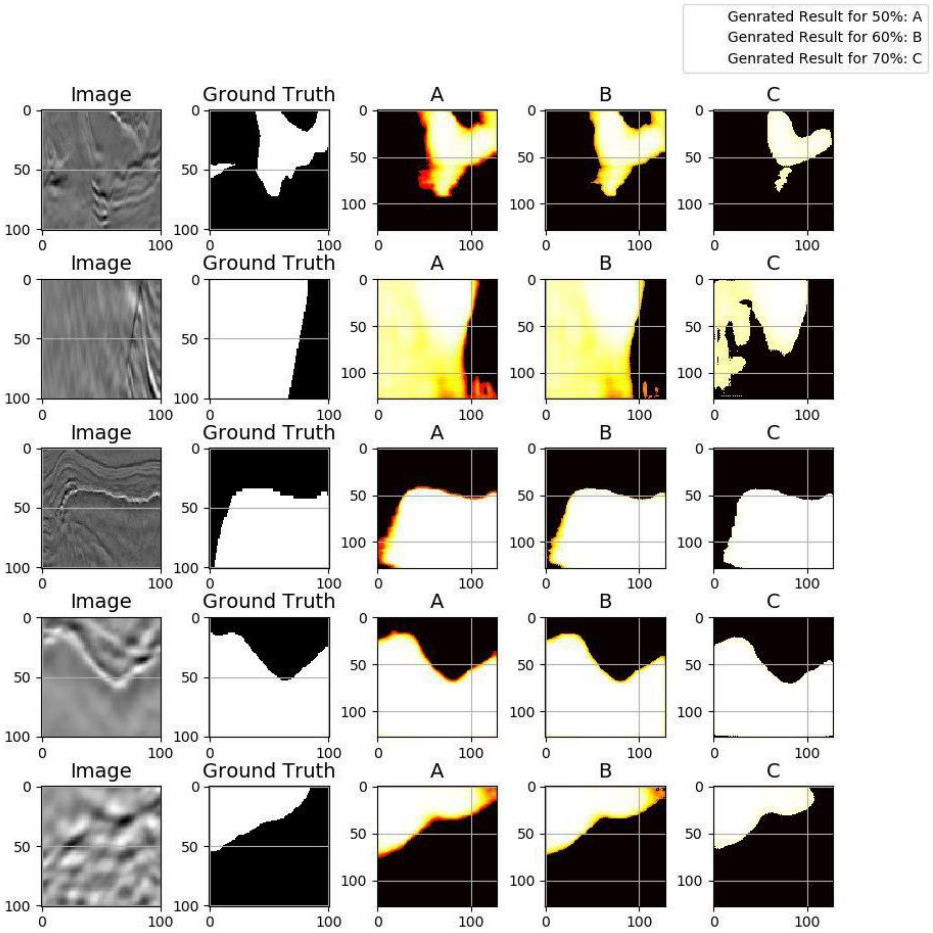


Figure 6.2.2.1: Comparison of Input, Ground Truth and Predicted Masks with 3 different thresholds.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

We present a novel approach to the challenging multistep seismic model-building problem. It uses a deep learning system to map out a salt body in the sub surface, using raw seismic recordings as input. A distinguishing aspect of the solution is the use of the U-Net model, which is suited to problems in semantic segmentation. We demonstrate the system's performance on real world data sets with complex salt body formations.

7.2 Future Work

The primary challenge going forward will be transitioning to salt bodies with more complex 3D geometry. This will lead to more tuned existing networks and/or extensions to our workflows. The application of machine learning approaches to seismic imaging and interpretation shows great promise in hydrocarbon exploration and can dramatically change how the vast amount of seismic data is used in the future.

REFERENCES

[1] Rastogi, R., 2011, High performance computing in seismic data processing: Promises and challenges: Presented at HPC Advisory Council Switzerland Workshop 2011, http://www.hpcadvisorycouncil.com/events/2011/switzerland_workshop/pdf/Presentations/Day%203/2_CDAC.pdf, accessed 16 January 2017.

[2] Shi, Yunzhi & wu, Xinming & Fomel, Sergey. (2018). Automatic salt-body classification using deep-convolutional neural network. 10.1190/segam2018-2997304.1.

[3] Guillen, Pablo & Larrazabal, German & Gonzalez, Gladys & Boumber, Dainis & Vilalta, Ricardo. (2015). Supervised learning to detect salt body.

[4] Frogner, C., C. Zhang, H. Mobahi, M. Araya-Polo, and T. A. Poggio, 2015, Learning with a Wasserstein loss: Presented at Advances in Neural Information Processing Systems (NIPS) 28.

[5] Araya-Polo, Mauricio, Taylor Dahlke, Charlie Frogner, Chiyuan Zhang, Tomaso Poggio, and Detlef Hohl. "Automated Fault Detection Without Seismic Processing." The Leading Edge 36, no. 3 (March 2017): 208–214 © 2017 Society of Exploration Geophysicists

[6] Khan, Asifullah. (2016). Introduction to Deep Convolutional Neural Networks.

[7] Ronneberger, Olaf & Fischer, Philipp & Brox, Thomas. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. LNCS. 9351. 234-241. 10.1007/978-3-319-24574-4_28.

[8] Dahlke, T., M. Araya-Polo, C. Zhang, and C. Frogner, 2016, Predicting geological features in 3D Seismic Data: Presented at Advances in Neural Information Processing Systems (NIPS) 29, 3D Deep Learning Workshop.

[9] Lin, H., and M. Tegmark, 2016, Why does deep and cheap learning work so well?: arXiv:1608.08225.