**CHAPTER1**

**INTRODUCTION**

1. **INTRODUCTION**

This Project is aimed to provide the most accurate seismic images and 3D renderings to classify and predict automatically and accurately, if a subsurface target is salt or not in order to analyze the presence of petroleum reserves beneath the surface of assessment.

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. The main motive of our system is to provide a highly feasible system to resolve the above described abnormalities.

**1.1 OBJECTIVE:**

To provide a salt identification application service for the companies which can be used anywhere and anytime., To minimize the man power using the application. Software application is developed mainly for Service of oil and gas company drillers and improve efficiency of work in a secured manner and provide fast service to solve problems. Any user can View Seismic information, Admin maintains all the salt content details by updating the records about seismic images and by also Viewing the production Requests, User can View Their Personal Account Information, about adding their collected images and User Can Check Their results once approve by the admin. This software has various advantages like fast processing, 24/7 access, high accuracy and ease of use.

**1.2PROJECT DESCRIPTION**

There are several areas of the world where there are vast quantities of salt in the subsurface. One of the challenges of seismic imaging is to identify the part of subsurface which is salt. Salt has characteristics that makes it both simple and hard to identify. **Salt density is usually 2.14 g/cc which is lower than most surrounding rocks**. The seismic **velocity of salt is 4.5 km/sec**, which is usually faster than its surrounding rocks. This difference creates a sharp reflection at the salt-sediment interface. Usually salt is an amorphous rock without much internal structure. This means that there is typically not much reflectivity inside the salt, unless there are sediments trapped inside it. The unusually high seismic velocity of salt can create problems with seismic imaging. Admin get the raw seismic image and passes it to the model, which produces the classified image with salt boundaries. These images can be analyzed further by employees to proceed mining for oil.

**1.3 SCOPE OF THE PROJECT**

* Data is maintained in a centralized database.
* Chances of losing data is very less.
* Easy to analyze Seismic images and classify salt boundaries
* Analysis and prediction of salt can be done with speed and precision.

**CHAPTER 2**

**LITERATURE SURVEY**

**[1]** **Twitter Sentiment Analysis using combined LSTM-CNN Models**

**Pedro M. Sosa -June 7, 2017**

The author of the paper describes how to apply deep learning techniques to implement a neural network model sentiment analysis from social media posts and messages. In this paper they have proposed 2 neural network models: CNN-LSTM and LSTM-CNN, which aim to combine CNN and LSTM networks to do sentiment analysis on Twitter data. They have provided a detailed explanation for both network architecture and perform comparisons against regular CNN, LSTM, and Feed-Forward networks. Finally, they found that both our models achieve better results, with best model (LSTM-CNN) achieving around 2.7%-8.5% better than regular models.

The contribution of this paper is to identify the evident based sentiment assessment. The methodologies that took place in this system are natural language processing for text mining, machine learning techniques. It shows an approach of assessing the online social media(twitter) content using supervised learning technique. This paper faced many challenges and most important challenge is improvising the accuracy of the content. They have used twitter media posts and hashtags as an primary factory and by using LSTM-CNN as their primary model. This paper came across different experiments using different classifiers trained on different combination of features were performed on the three aforementioned data sets. Because of the large size of the data sets, roughly 20% of each corpus were saved untouched to be used as test sets. The remaining 80% of the data sets were used as training sets. The paper contributes to our proposed system by the method of how the deep learning technique work fine with the information provided and classification process is achieved by LSTM-CNN model and about the automating the assessment process.

**[2]** **EFFECTIVENESS OF U-NET IN DE-NOISING RGB IMAGES, by Rina Komatsu and Tad Gonsalves**

**Department of Information & Communication Sciences, Faculty of Science & Technology, Sophia University, Tokyo, Japan – August 2019**

The author of the paper describes about how different classifier deals with large amount of datasets and de noising RBG images to refine it to high resolution .The paper contributes to our proposed system by the way to manage the large dataset to feed to the machine and the architecture used to implement the model. In this paper, large number of works is carried out to figure of efficient method of Denoising images of RGB. Digital images often contain “noise” which takes away their clarity and sharpness. Most of the existing denoising algorithms do not offer the best solution because there are difficulties such as removing strong noise while leaving the features and other details of the image intact. Faced with the problem of denoising, the author tried solving it with a Convolutional Neural Network architecture called the “U-Net”. This paper deals with the training of a U-Net to remove 3 different kinds of noise: Gaussian, Blackness, and Camera shake. The systems result indicate the effectiveness of U-Net in denoising images while leaving their features and other details intact.

To remove various noises no matter how strong, technology like the human brain which can distinguish between noise and the original image is needed. In recent years, AI deep learning algorithms and techniques are rapidly progressing to handle recognition, segmentation, re- production of images, etc. Using deep learning techniques, the author investigated if it is possible to build a model that can effectively denoise noisy input images and get clear output images without any traces of noise. In this paper, they picked up the CNN architecture “U-Net” as a typical denoising deep learning model and developed their own “Reformed U-Net” to handle strong noise in the images. In Section 2 of the paper they have introduced 3 kinds of noises we used as denoising targets. Section 3 gives a brief information about the U-Net overview and introduces comparative studies of the 2 models based on U-Net. Section 4 and 5 describes the programming environment and experimental setup. Section 6 indicates denoising results. Section 7 concludes this study.

**[3] The Effectiveness of Data Augmentation in Image Classification using Deep Learning**

**Jason Wang -** **Stanford University 450 Serra Mall zwang01@stanford.edu Luis Perez Google -** **1600 Amphitheatre Parkway**

In this paper, The author Explore's and compare’s multiple solutions to the problem of data augmentation in image classification. Previous work’s in image augmentation has demonstrated the effectiveness of data augmentation through simple techniques, such as cropping, rotating, and flipping input images. We artificially constrain our access to data to a small subset of the ImageNet dataset, and compare each data augmentation technique in turn. One of the more successful data augmentations strategies is the traditional transformations mentioned above. We also experiment with GANs to generate images of different styles. Finally, we propose a method to allow a neural net to learn augmentations that best improve the classifier, which we call neural augmentation. We discuss the successes and shortcomings of this method on various datasets. Dataset includes GTSRDC Road traffic signs dataset provided by traffic cooperation department of the German country.

**[4] REVIEW ON IMAGE SEGMENTATION TECHNIQUES**

**Associate Professor Dr.S.Kannan**

**Department of Computer Applications,**

**Madurai Kamaraj University.**

**Vairaprakash Gurusamy,**

**Research Scholar,**

**Department of Computer Applications,**

**Madurai Kamaraj University.**

Digital image processing supports a strong research program in areas of image enhancement and image-based pattern recognition. Among the various image processing techniques image segmentation plays a vital role in step to analyze the given image. Image segmentation is the fundamental step to analyze images and extract data from them. This work deals on the basic principles on the methods used to segment an image. Segmentation has become a prominent objective in image analysis and computer vision. To segment the images, from segmentation techniques edge detection, thresholding, region growing and clustering are taken for this study. Segmentation algorithms are based on two properties similarity and discontinuity. This paper focuses on the various methods that are widely used to segment the image.

**[5] Fully Convolutional Networks for Semantic Segmentation**

**Jonathan Long**

**Evan Shelhamer**

**Trevor Darrell**

**UC Berkeley {jonlong,shelhamer,trevor}@cs.berkeley.edu.**

Convolutional networks are powerful visual models that yield hierarchies of features. We show that convolutional networks by themselves, trained end-to-end, pixels-to-pixels, exceed the state-of-the-art in semantic segmentation. Our key insight is to build “fully convolutional” networks that take input of arbitrary size and produce correspondingly sized output with efficient inference and learning. We define and detail the space of fully convolutional networks, explain their application to spatially dense prediction tasks, and draw connections to prior models. We adapt contemporary classification networks (AlexNet [22], the VGG net [34], and GoogleLet ional networks and transfer their learned representations by fine-tuning [5] to the segmentation task. We then define a skip architecture that combines semantic information from a deep, coarse layer with appearance information from as hallow, fine layer to produce accurate and detailed segmentations. Our fully convolutional network achieves state-of-the-art segmentation of PASCAL VOC (20% relative improvement to 62.2% mean IU on 2012), NYUDv2, and SIFTFlow, while inference takes less than one fifth of a second for a typical image.

**CHAPTER 3**

**SYSTEM ANALYSIS**

**3.1. EXISTING SYSTEM**

Seismic data is collected using reflection seismology, or seismic reflection. The method requires a controlled seismic source of energy, such as compressed air or a seismic vibrator, and sensors record the reflection from rock interfaces within the subsurface. The recorded data is then processed to create a 3D view of earth’s interior. Reflection seismology is similar to X-ray, sonar and echolocation. A seismic image is produced from imaging the reflection coming from rock boundaries. The seismic image shows the boundaries between different rock types. In theory, the strength of reflection is directly proportional to the difference in the physical properties on either side of the interface. While seismic images show rock boundaries, they don't say much about the rock themselves, some rocks are easy to identify while some are difficult.

There are several areas of the world where there are vast quantities of salt in the subsurface. One of the challenges of seismic imaging is to identify the part of subsurface which is salt. Salt has characteristics that makes it both simple and hard to identify. Salt density is usually 2.14 g/cc which is lower than most surrounding rocks. The seismic velocity of salt is 4.5 km/sec, which is usually faster than its surrounding rocks. This difference creates a sharp reflection at the salt-sediment interface. Usually salt is an amorphous rock without much internal structure. This means that there is typically not much reflectivity inside the salt, unless there are sediments trapped inside it. The unusually high seismic velocity of salt can create problems with seismic imaging.

Thus, to create a solution, we propose a model based on U-NET architecture of Convolutional Neural Network. This model takes in a seismic image as input and converts it into gray scale image and starts to process it. Following the hierarchical order of U\_NET of resolving an image from high dimension to a smaller dimension, processing the image to classify according to the training set provided and segmenting the salts boundary from the given input image and displaying it in image format.

**3.1.1. Disadvantages of Existing System**

* Lack of accuracy in derived data.
* More man power required on the field to collect images.
* Time consuming.
* Consumes a large volume of People work.
* Needs manual calculations.
* No direct role for the higher officials.
* High diversity and error in assessment of the image for salt.

**3.2 Proposed System:**  
The aim of proposed system is to develop a system of improved facilities and accuracy in predicting the salt content in the seismic image. The proposed system can overcome all the limitations of the existing system. The system provides high accuracy, low calculations and reduces the manual work.

**3.2.1 ADVANTAGES OF PROPOSED SYSTEM:**

* High usage of limited information from the seismic image.
* Ensure data accuracies.
* Proper control of the higher parameters for measurement.
* Minimize manual data entry.
* Minimum time needed for the various processing.
* Greater efficiency.
* Better service.
* User friendliness and interactive.
* Minimum time required.

**3.3 REQUIREMENT SPECIFICATION:**

**3.3.1 HARDWARE REQUIREMENTS:**

* Hard Disk: 40GB and above
* Cloud drive 30GB and above
* Web browser stable release of any distribution
* RAM: 512MB and above
* Processor: Pentium IV and above

**3.3.2 SOFTWARE REQUIREMENTS:**

* Windows operating system 8 and above
* Python Version 3.0.0 and above
* Open CV version 3.4.1 and above
* Google-Colab or any other online platform to execute the code

**3.4 LANGUAGE SPECIFICATION:**

**3.4.1 PYHTON 3.0.0 and above:**

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 aim 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 in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 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.

The Python 2 language, i.e. Python 2.7.x, was officially discontinued on 1 January 2020 (first planned for 2015) after which security patches and other improvements will not be released for it. With Python 2's end-of-life, only Python 3.5.x and later are supported.

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

**3.4.2 Open CV:**

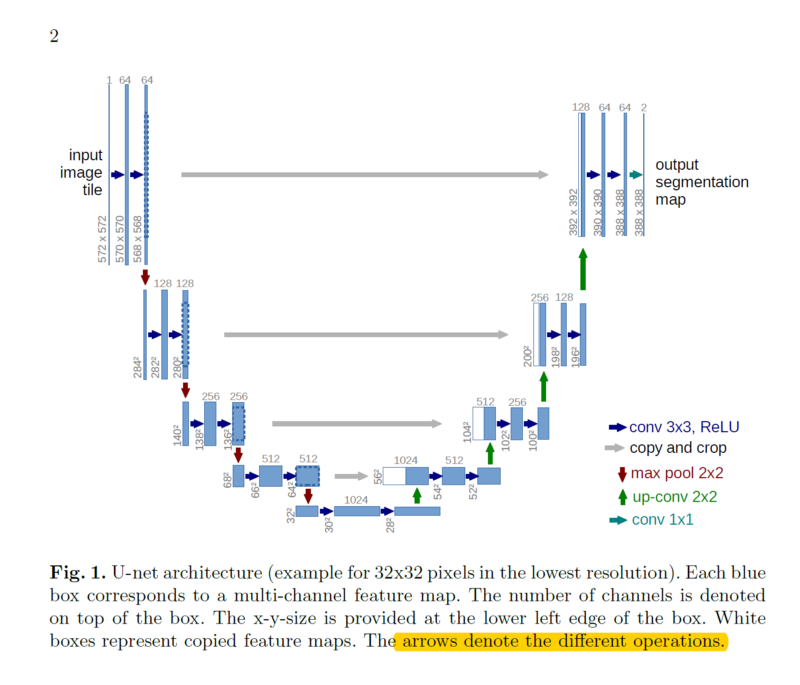
**OpenCV** (*Open source computer vision*) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.

OpenCV supports some models from deep learning frameworks like TensorFlow, Torch, PyTorch (after converting to an ONNX model) and Caffe according to a defined list of supported layers.

**CHAPTER 4**

**SYSTEM DESIGN**

**4.1.1 SYSTEM ARCHITECTURE**  
**Introduction**

The system architecture consists of a basic u-net architecture coupled with a convolutional neural network model. 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 an image of any size.

**FIG.NO: 4.1.1 U-NET ARCHITECTURE**.

**Input design**

The design of input focuses on controlling the image provided to the u-net model. The input is a seismic image extracted from seismic imaging technique. this image is further passed into the u-net block which in turn access multiple unit CNN blocks into the system. These units process the image as described in the architecture. The image is down-sampled from a high resolution of 255 \* 255 to a unit cell block. The block is analyzed and processed in that format for simplicity and to reduce time complexity

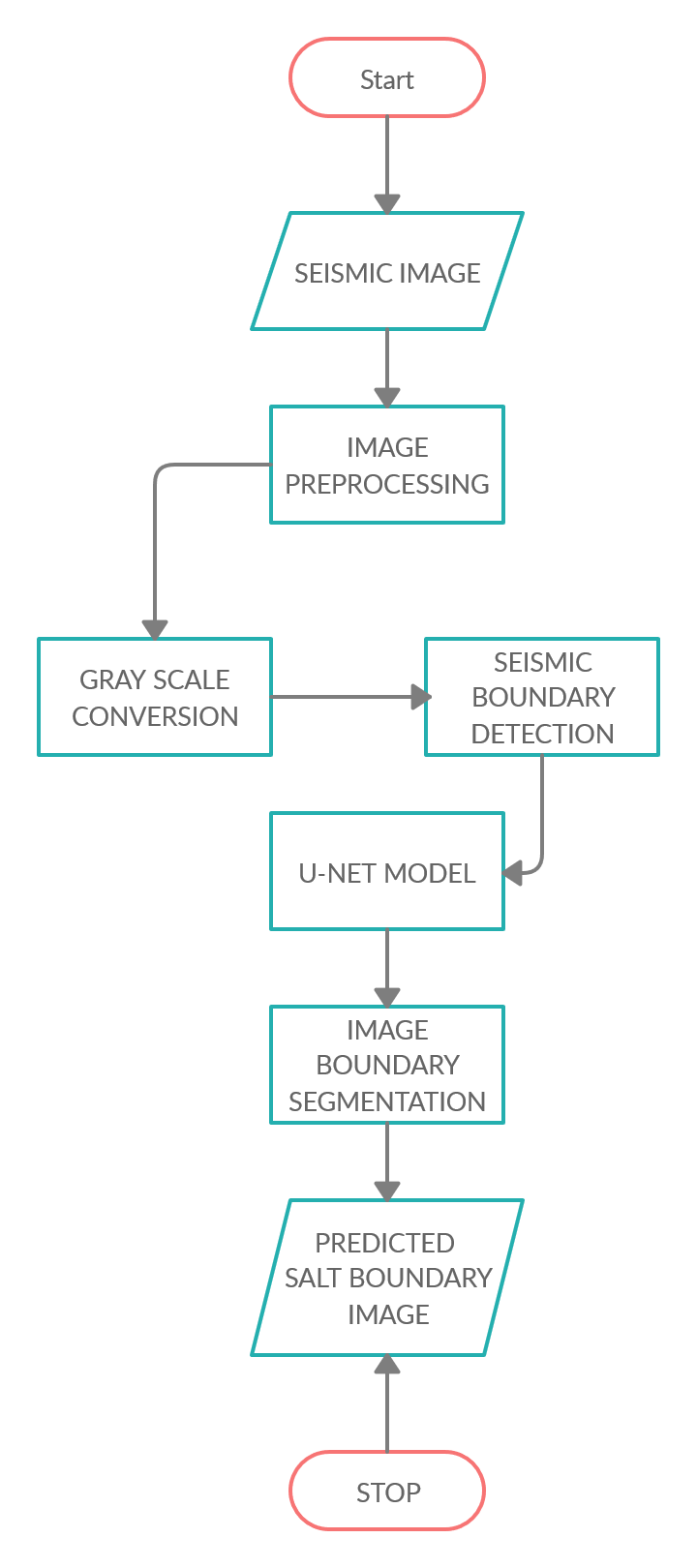
**Output Design**

A quality output is one, which meets the requirements of the end user and presents the information clearly. In output is design in such a way that the accuracy of the model is improvised for each iteration of model training. The down-sampled image of resolution dimension 1 \* 1 is up-sampled again after processing in its unit size.

The image is further classified on basis of the training data provided and the salts boundary is drawn. The depicted output is displayed with an inbuild visualizing technique provided by the processing programming language itself. The boundary splits the salt and no salt region by image instance segmentation.

The process takes place in the following order:

1. Parameter detection.
2. Semantic segmentation.
3. Instance segmentation.
4. Boundary classification.

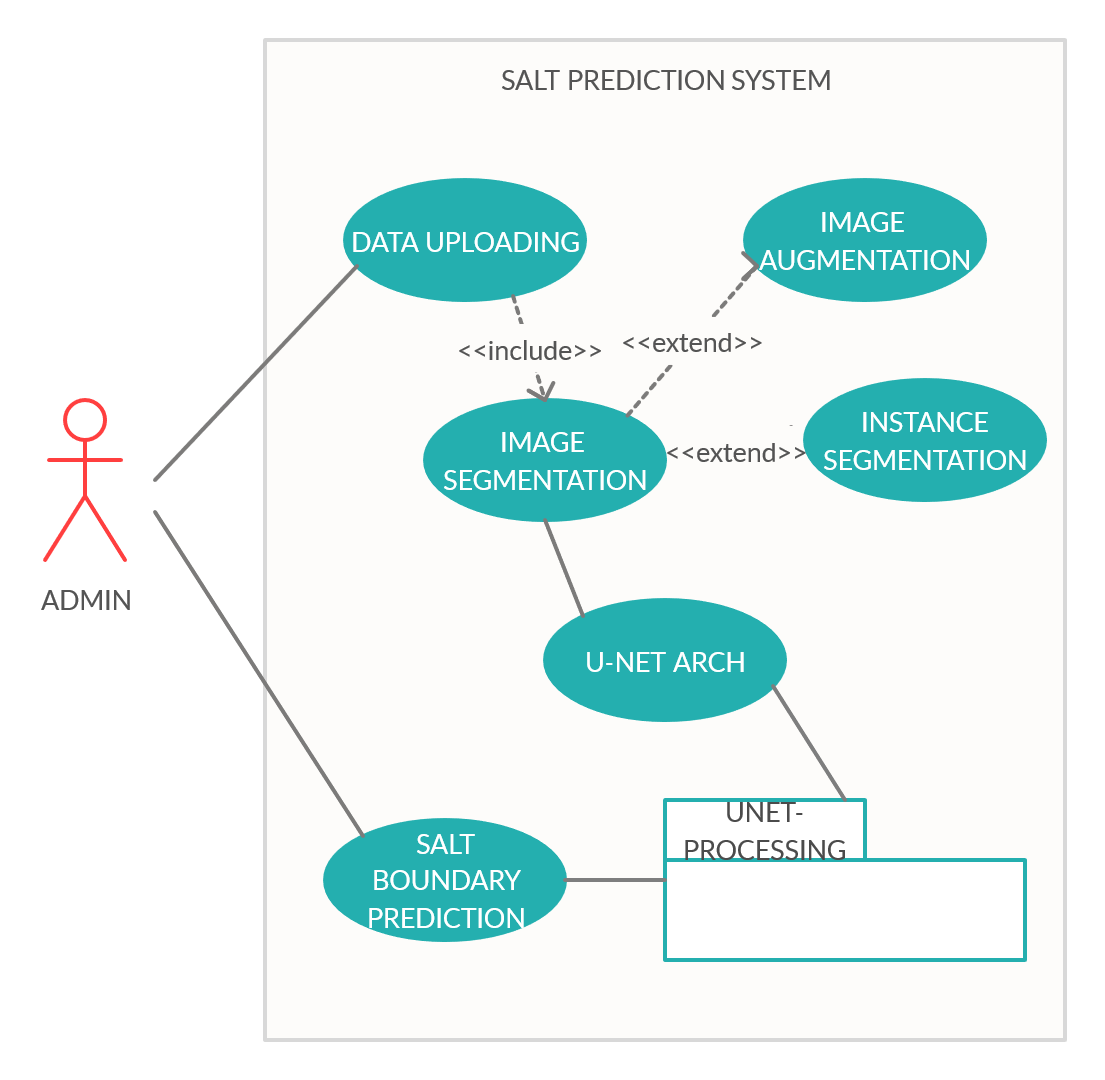


**Fig 4.1.1.2 System Architecture**

**4.2 UML DIAGRAMS**

**4.1.2 USE CASE DIAGRAM**

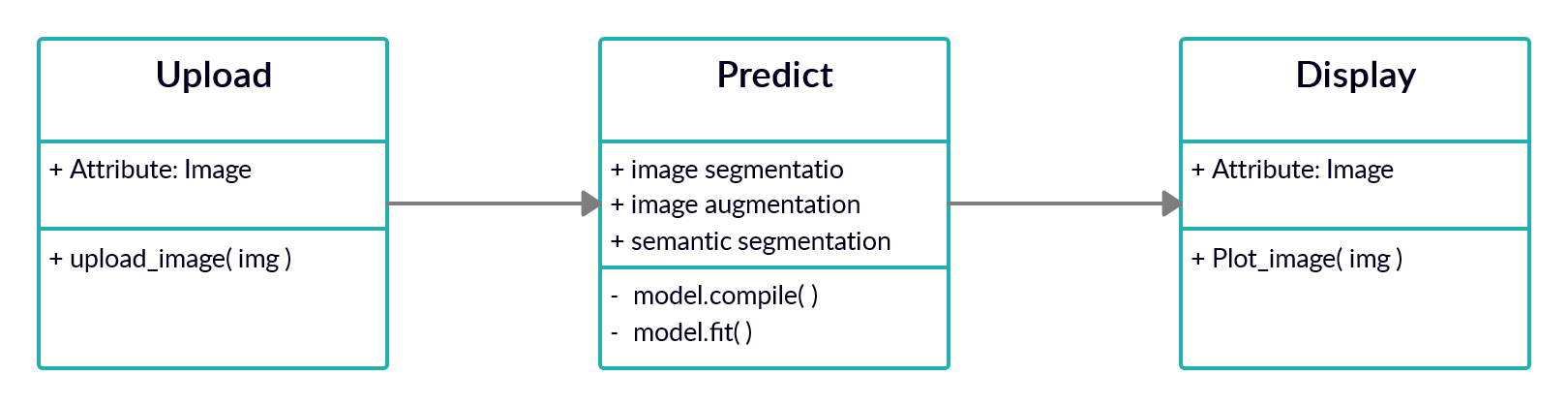
In the given figure 2.3.1.1, the diagram depicts how the system would function. The admin uploads data into the system, the system in tun processes the image in three different segmentation techniques and pushes it to the u-net model which is pre-trained. The u-net package processes the image, segments the salt boundary and plots it using in built packages from the programming language.



**Fig 4.1.2.Use Case Diagram**

**4.1.3 CLASS DIAGRAM**

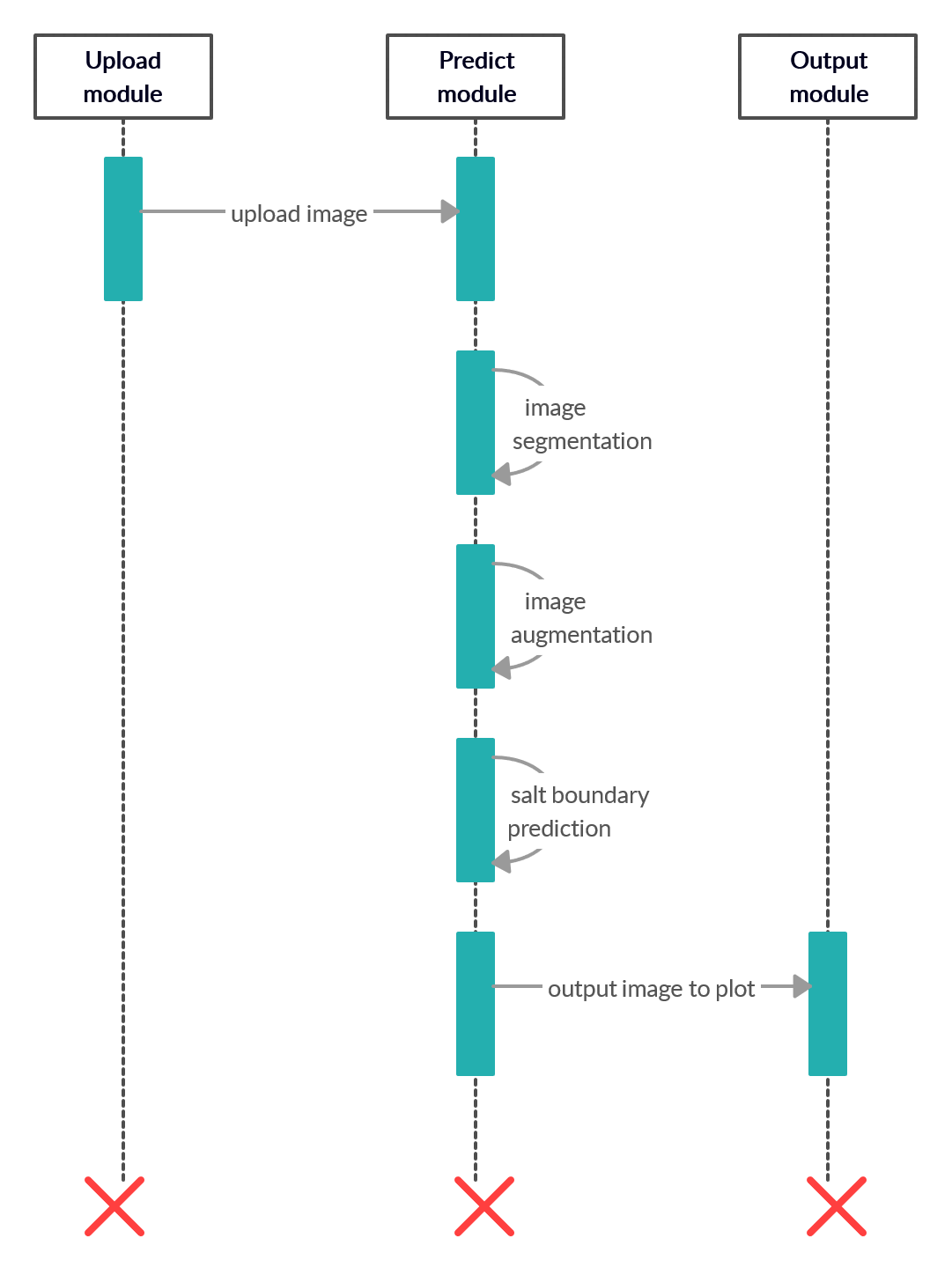
The class diagram of salt prediction system consists of 3 main classes. The upload class is used to get a seismic image input from the admin and pass it to the main model for computations. The predict or run class is used to perform 3 different image processing techniques in order to enhance the image passed into the predictive model. Then the prediction of boundary takes place in it and it is passed to next model. The final class is the Display class where it takes the output image from the Predict class and displays the various plotted images in the static view. These images include seismic image, grey scale image, boundary value image.



**Fig 4.1.3 Class Diagram**

**4.1.4.SEQUENCE DIAGRAM**

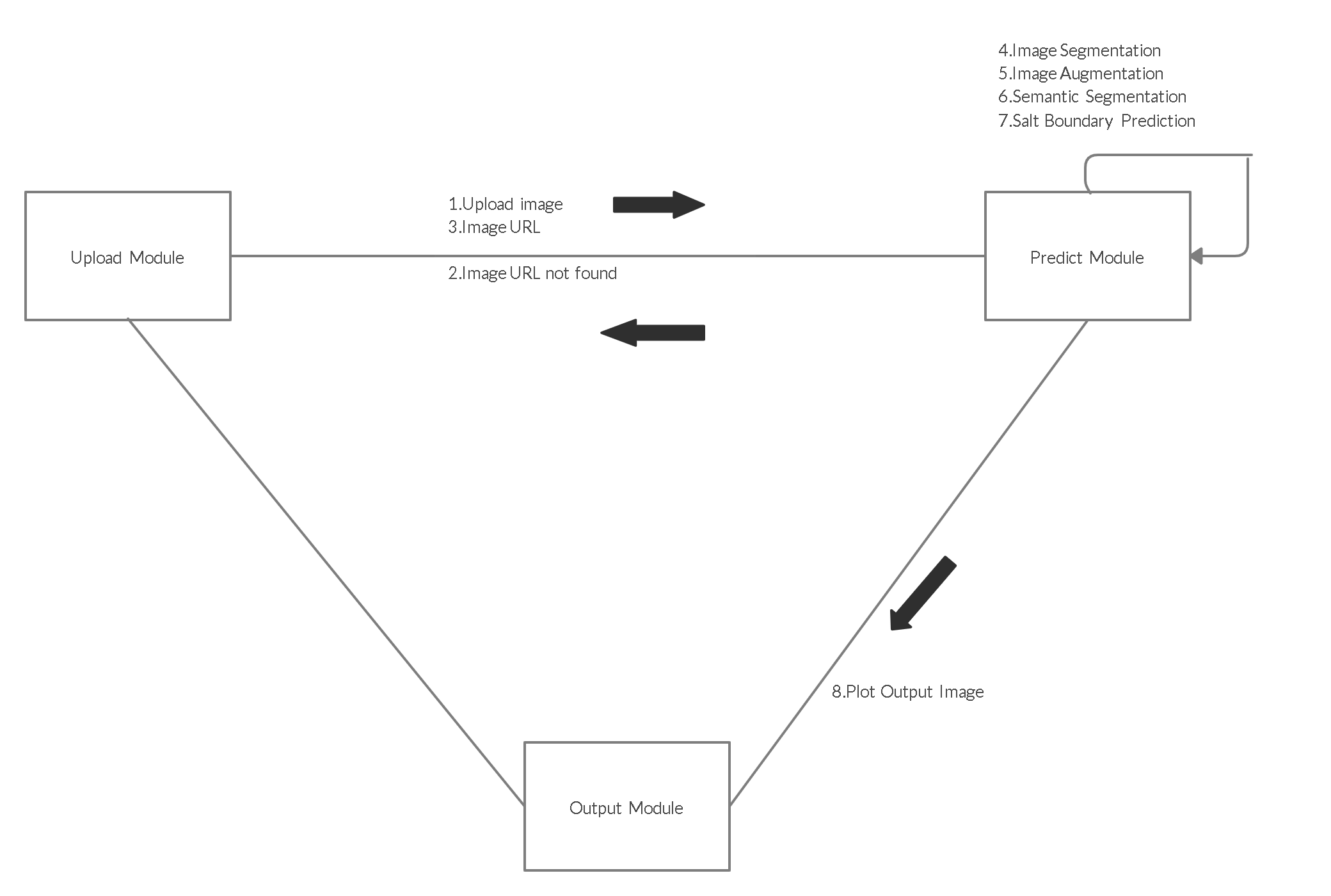
The sequence diagram depicts the sequence of operations that takes place to produce a valid output. The module starts by getting an input seismic image from the user in the upload module. This image is parsed into the predict module where the image is segmented, augmented, and sent to predict the boundary of the salt. The final predicted module is sent to the output module, where an plot function is used to display the result along with the image of salt boundary.



**Fig. 4.1.4 Sequence Diagram**

**4.1.5.COLLABORATION DIAGRAM:**

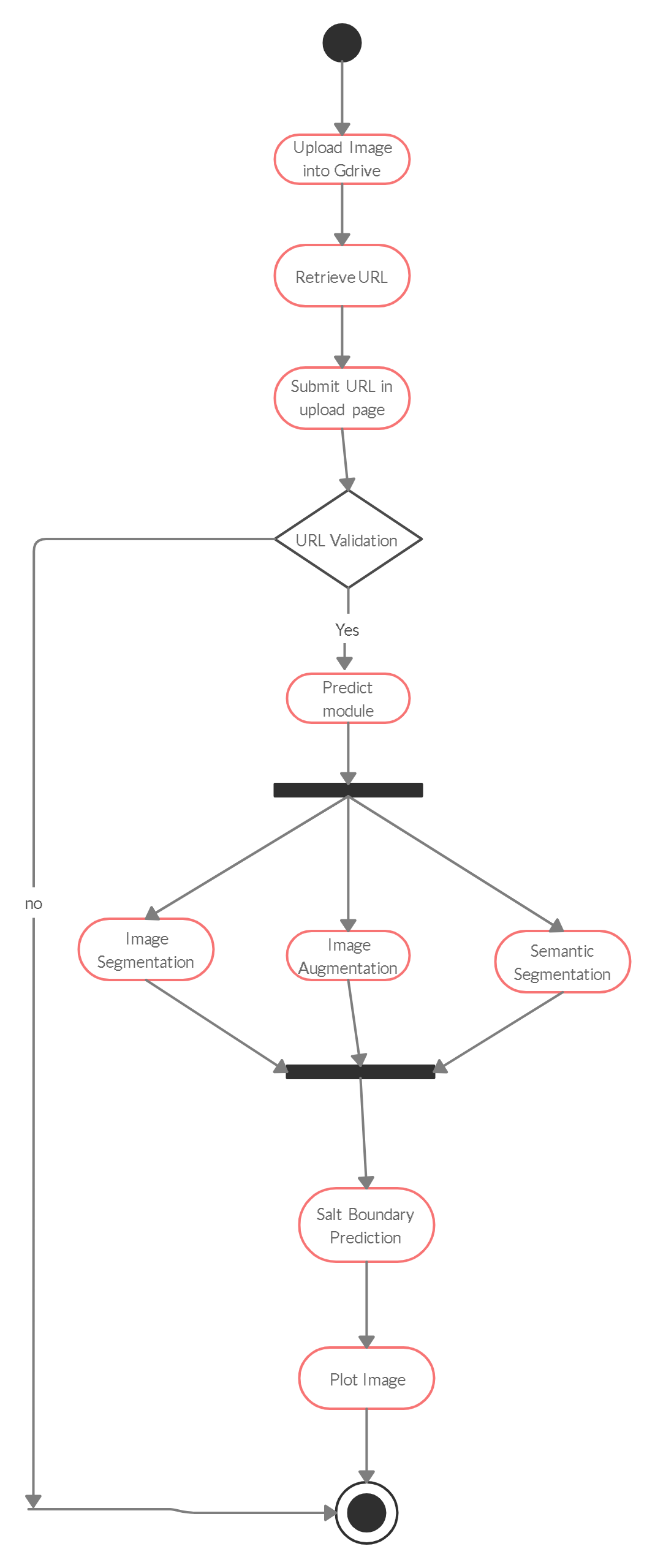
The upload module sends a upload request with URL of image in G-drive to predict module. On the way to it, it is analysed for a valid URL or not. If ok it is sent to the system. If not, it is discarded. The predict module would communicate within itself with 3 different image processing processes. It later sends the processed data to display module where the analysed data with its accuracy is displayed. The displayed image consists of boundary of salt along with the content of salt in it.



**Fig 4.1.5 Collaboration Diagram**

**4.1.6.ACTIVITY DIAGRAM**

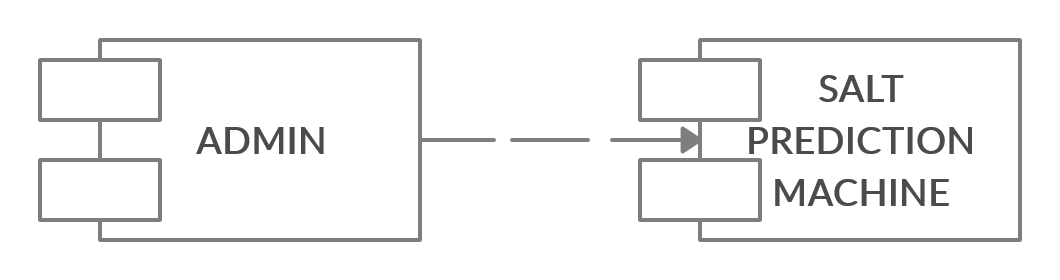
Activity diagram of the salt prediction system is described as follows. The admin uses his portal to upload an image into his specified G-drive of a customized account. Then the URL of the image is retrieved by him/her and placed in the URL upload section in the admin page. The URL is later validated to check if it’s a valid URL or not. If its valid it flows to next step, else it is discarded and session starts as new. Once the image from the URL is retrieved, it is sent to 3 different modules of image segmentation, image augmentation and image semantic segmentation. All the outputs of 3 modules are collaborated and sent to the predict module. The predict module contains the core component of u-net architecture with static convolutional neural network blocks which are called into the architecture. Then the image is processed and the boundary of salt is predicted and plotted. The plotted output with all other processed image is displayed. The accuracy of the model along with is loss data value is also plotted and displayed in the pre-output page



**Fig 4.1.6 Activity Diagram**

**4.1.7. COMPONENT DIAGRAM**

Component diagram of this salt prediction system consists of 2 main, high priority components for proper function. The admin uploads a seismic image into the machine or software and it computes the boundary of salt present in the image. The salt prediction machine is a unique which is developed with the necessary modules such as the image segmentation, augmentation and predictions. This is the core module or component of the entire system



**Fig 4.1.7 Component Diagram**

**4.1.8. DEPLOYMENT DIAGRAM**

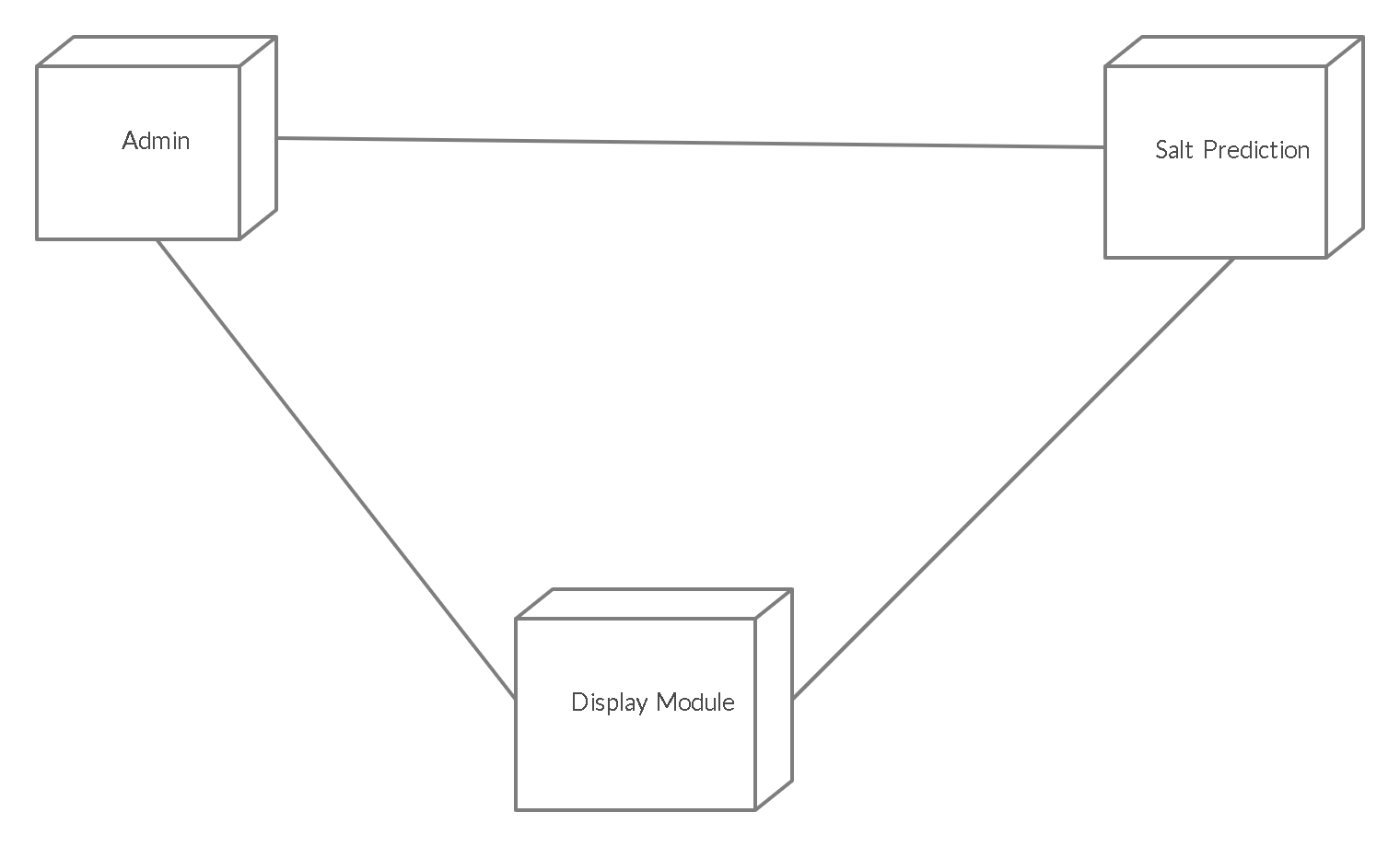
The deployment diagram in the [Unified Modeling Language](http://en.wikipedia.org/wiki/Unified_Modeling_Language) models the physical deployment of artefacts on [nodes](http://en.wikipedia.org/wiki/Node_(UML)). To describe the model for salt prediction, the diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server[Google G-Drive]), what software components ("artefact's") run on each node (e.g., web application, database, Deep Learning based U-NET architecture model), and how the different pieces are connected (e.g. JDBC, REST, RMI,mage-API).

The nodes appear as boxes, and the artefacts allocated to each node appear as rectangles within the boxes. Nodes may have sub nodes, which appear as nested boxes. A single node in a deployment diagram may conceptually represent multiple physical nodes, such as a cluster of database servers.

Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed. So deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships

The purpose of deployment diagrams can be described as:

* Visualize hardware topology of a system.
* Describe the hardware components used to deploy software components.
* Describe runtime processing nodes.



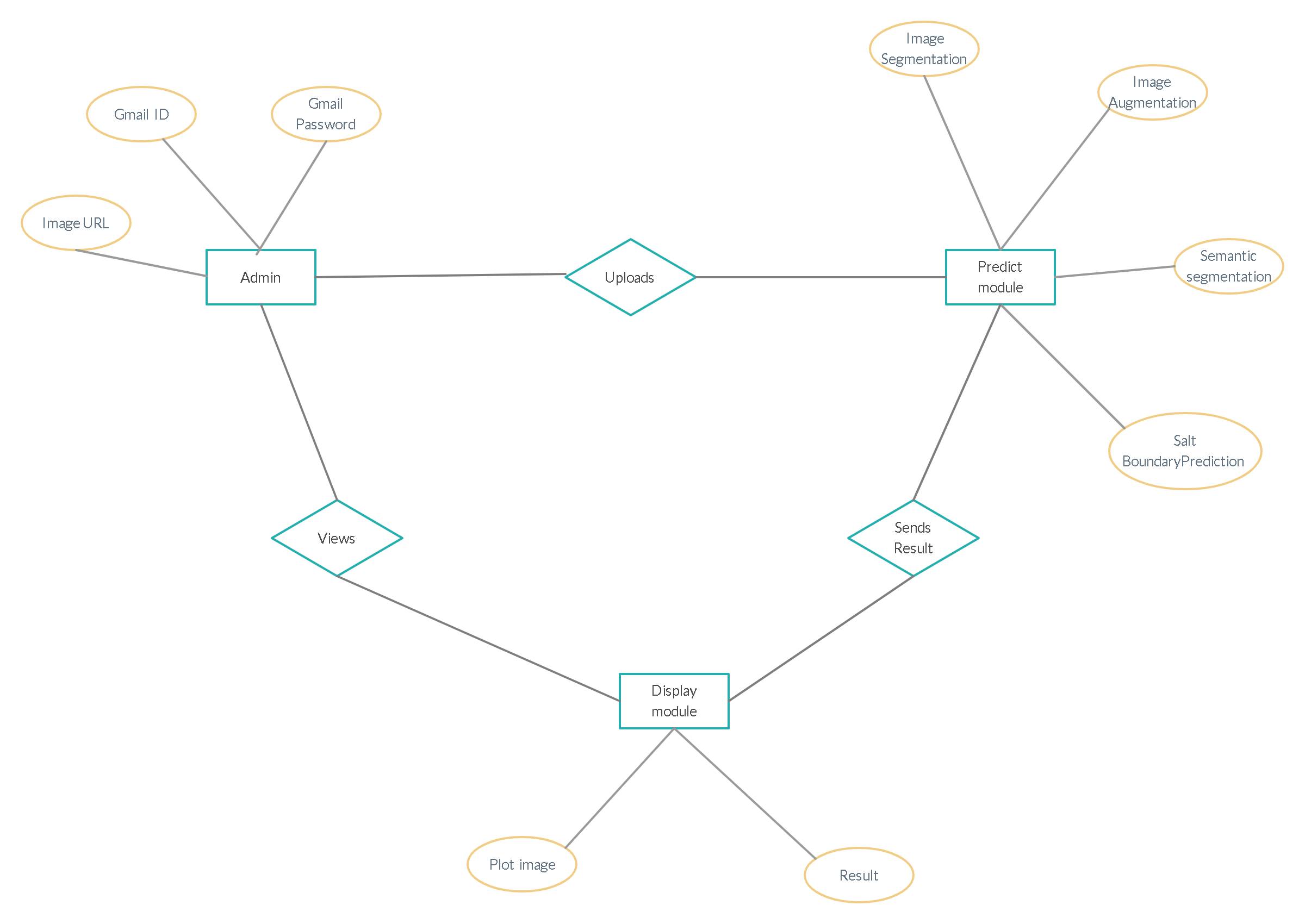
**Fig 4.1.8 Deployment Diagram**

**4.1.9 ER Diagram**

An entity–relationship model (ER model) is a [data model](http://en.wikipedia.org/wiki/Data_modeling) for describing the data or information aspects of a business domain or its process requirements, in an abstract way that lends itself to ultimately being implemented in a [database](http://en.wikipedia.org/wiki/Database) such as a [relational database](http://en.wikipedia.org/wiki/Relational_database). The main components of ER models are [entities](http://en.wikipedia.org/wiki/Entities) (things) and the relationships that can exist among them.

An entity-relationship model is a systematic way of describing and defining a business process. The process is modeled as components (entities) that are linked with each other by relationships that express the dependencies and requirements between them, such as: one building may be divided into zero or more apartments, but one apartment can only be located in one building. Entities may have various properties (attributes) that characterize them. Diagrams created to represent these entities, attributes, and relationships graphically are called entity–relationship diagrams.

Thus, the ER diagram of the salt prediction system has 3 entities. The entity one has 3 attributes - image URL, Gmail-id, Gmail-password. It is further connected to predict entity with an “uploads” relationship. The predict entity has 4 attributes – image segmentation, image augmentation, semantic segmentation, salt boundary prediction. It is connected to Display module via a “sends result” relationship. The display entity has 2 attribute's – Plot image, Result. It is connected to Admin entity with “views” relationship as the admin is allowed to view the result in the display page.



**Fig 4.1.9 Entity Diagram**

**CHAPTER 5**

**MODULES**

**5.MODULES**

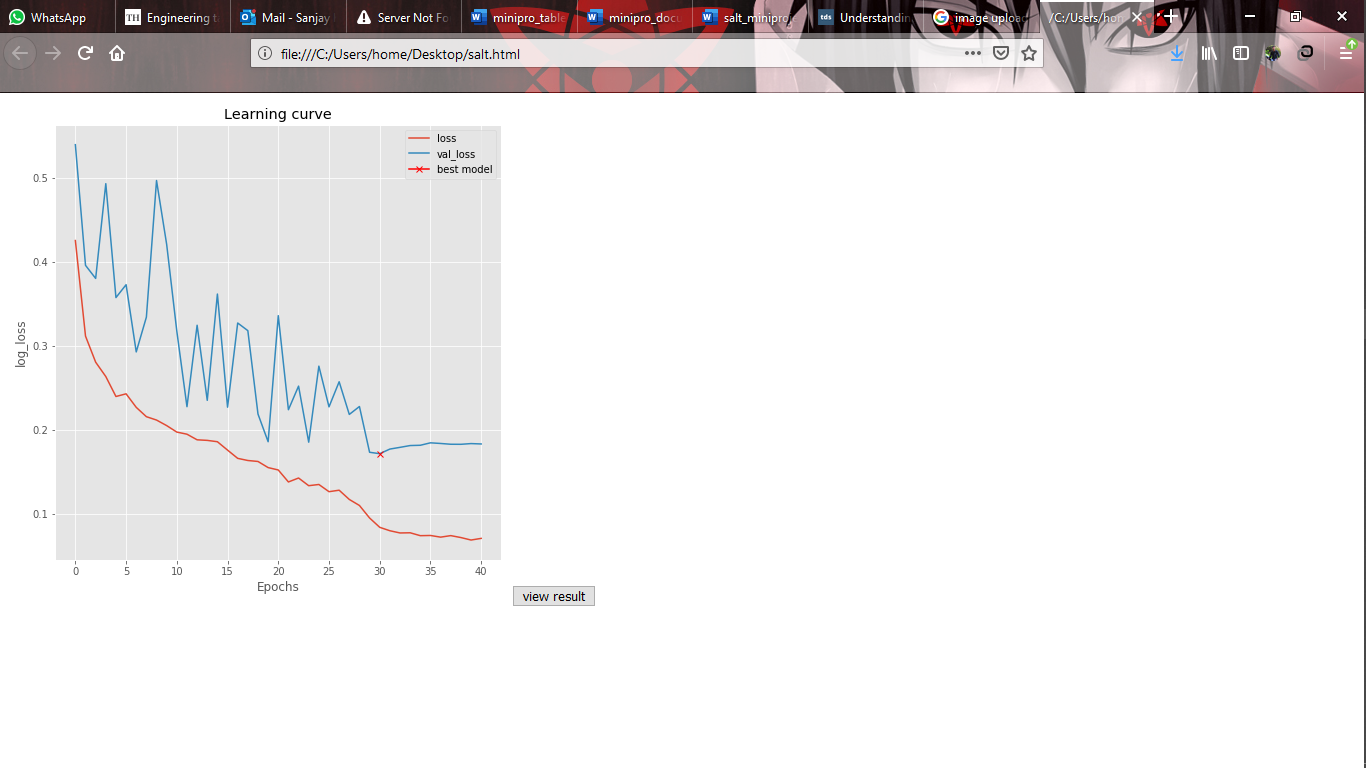
* Admin image input
* Predict module
* Display module.
  1. **MODULES DESCRIPTION:**

**5.1.1 Admin image upload Page**

Admin can upload a seismic image URL into the remote file server designed for the special functioning of our computed system. The seismic image is first uploaded into a drive folder of google, customized and provided for admin. Once the image URL is uploaded, it prevails in the cache data of the web-server or the local server. If the time exceeds the threshold value set for it, the image URL will be discarded. If not, the image is retrieved, is processed and sent to the predict module or the computational page.

**5.1.2 Predict module:**

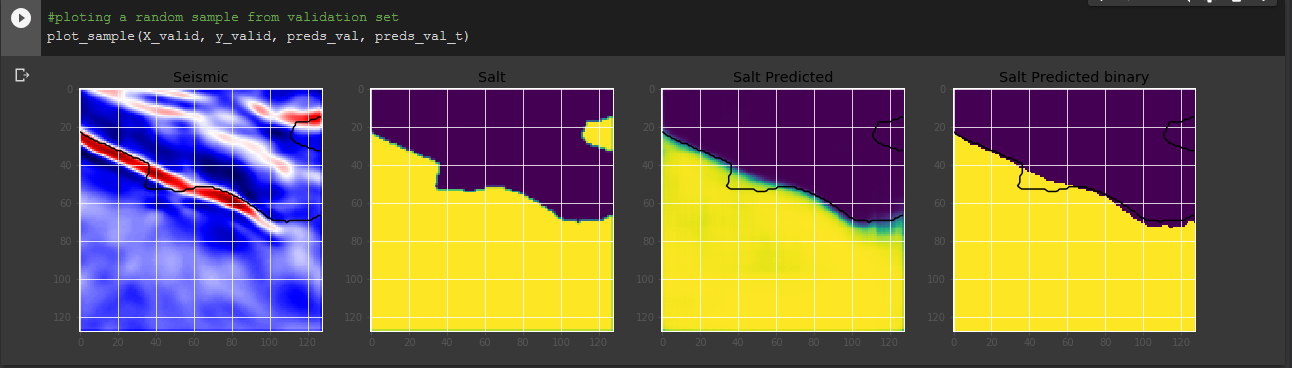
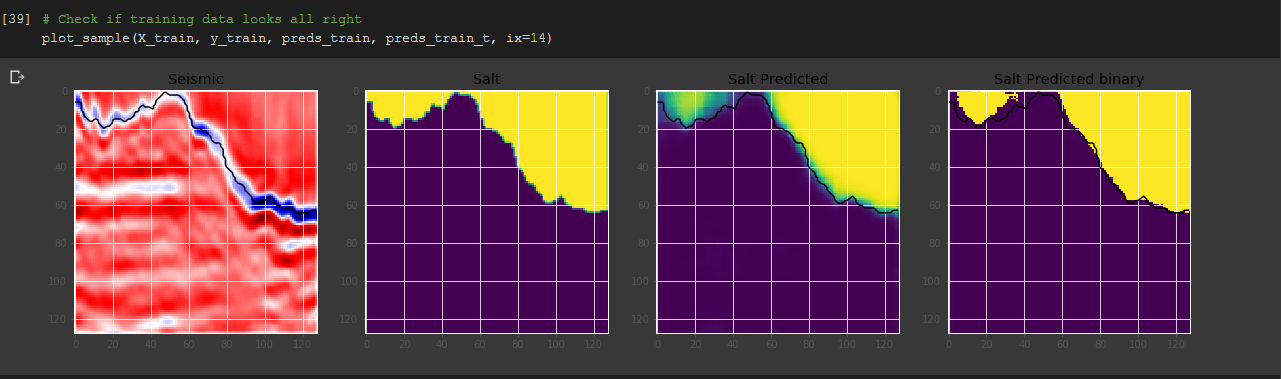
This module helps to retrieve image from the storage bucket and send it into the processing module. The image is sent into 3 different image processing sub modules inside it. The image augmentation module enumerates this single image into multiple image of different dimensions and filters. The image semantic segmentation separates the anomaly(salt) which is trained in the model. The image instance segmentation classifies the salt bodies boundary, the not part and processes it into an image and stores it in modules cache memory.



**Fig 5.1.2 Accuracy display**

**5.1.3 Display Result page:**

Admin can view the result of the seismic image as soon as he views the validation loss of the module. The image below is provided for 2 different seismic images provided with the link from g-drive. The image displayed is provided with the seismic image in the left with, salt, salt predicted, salt predicted binary. In the provided image, the image with red and blue is the seismic image data with high chances of misprediction. The image with purple and yellow is used to present the salt content present in the seismic data image. The yellow color in the image represents the presence of salt , purple represents no presence of salt.



**Fig. 5.1.3 Result Display**

**CHAPTER 6**

**CONCLUSION AND FUTURE ENHANCEMENTS**

**6.1.1. CONCLUSION**

The salt identification and classification system initiate the objective of providing the user with customized and powerful processing of seismic images with high accuracy and process is speeded up exponentially. The software is built with a single operation, of loading a seismic image and extracting highly accurate salt boundry and content of salt present.

All the requirements specified during the analysis and design phase are fully met, thus resulting in the formation of good classification software. The interface provided is very user friendly and flexible for all times. The model built is highly accurate than any other existing model and thus helps maintaining the mining companies' safety both financially and we as man power wise.

**6.1.2. FUTURE ENHANCEMENT:**

Thorough this model is highly accurate, its accuracy can be enhanced by adding more images to the current dataset. The output provided by the current model only describes and produces the boundary of salt content. It can be enhanced to provide the amount of salt present in the particular area where the seismic image was captured. Thus, it forms a complete system to help the mining company from scratch to complete project assessment.

**APPENDIX**

**APPENDIX-1:**

**SAMPLE CODING:**

**5.1 Salt.py:**

**#import basic packages**

import os

import random

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

plt.style.use("ggplot")

%matplotlib inline

**#import image processing packages**

from tqdm import tqdm\_notebook, tnrange

from itertools import chain

from skimage.io import imread, imshow, concatenate\_images

from skimage.transform import resize

from skimage.morphology import label

from sklearn.model\_selection import train\_test\_split

import tensorflow as tf

**#import deep learning Tensor-flow packages**

from keras.models import Model, load\_model

from keras.layers import Input, BatchNormalization, Activation, Dense, Dropout

from keras.layers.core import Lambda, RepeatVector, Reshape

from keras.layers.convolutional import Conv2D, Conv2DTranspose

from keras.layers.pooling import MaxPooling2D, GlobalMaxPool2D

from keras.layers.merge import concatenate, add

from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau

from keras.optimizers import Adam

from keras.preprocessing.image import ImageDataGenerator, array\_to\_img, img\_to\_array, load\_img

**# Set some parameters image width ,height , border value**

im\_width = 128

im\_height = 128

border = 5

**# list of names all images in the given path**

ids = next(os.walk("images"))[2]

print("No. of images = ", len(ids))

X = np.zeros((len(ids), im\_height, im\_width, 1), dtype=np.float32)

y = np.zeros((len(ids), im\_height, im\_width, 1), dtype=np.float32)

**# tqdm is used to display the progress bar**

for n, id\_ in tqdm\_notebook(enumerate(ids), total=len(ids)):

**# Load images**

img = load\_img("images/"+id\_, color\_mode="grayscale")

x\_img = img\_to\_array(img)

x\_img = resize(x\_img, (128, 128, 1), mode = 'constant', preserve\_range = True)

**# Load masks**

mask = img\_to\_array(load\_img("masks/"+id\_, color\_mode="grayscale"))

mask = resize(mask, (128, 128, 1), mode = 'constant', preserve\_range = True)

**# Save images**

X[n] = x\_img/255.0

y[n] = mask/255.0

**# Split train and valid**

X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X, y, test\_size=0.1, random\_state=42)

**# Visualize any randome image along with the mask**

ix = random.randint(0, len(X\_train))

has\_mask = y\_train[ix].max() > 0 # salt indicator

fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (20, 15))

ax1.imshow(X\_train[ix, ..., 0], cmap = 'seismic', interpolation = 'bilinear')

if has\_mask: **# if salt**

**# draw a boundary(contour) in the original image separating salt and non-salt areas**

ax1.contour(y\_train[ix].squeeze(), colors = 'k', linewidths = 5, levels = [0.5])

ax1.set\_title('Seismic')

ax2.imshow(y\_train[ix].squeeze(), cmap = 'gray', interpolation = 'bilinear')

ax2.set\_title('Salt')

**#static conv2D code with batchnormalization.**

def conv2d\_block(input\_tensor, n\_filters, kernel\_size = 3, batchnorm = True):

"""Function to add 2 convolutional layers with the parameters passed to it"""

# first layer

x = Conv2D(filters = n\_filters, kernel\_size = (kernel\_size, kernel\_size),\

kernel\_initializer = 'he\_normal', padding = 'same')(input\_tensor)

if batchnorm:

x = BatchNormalization()(x)

x = Activation('relu')(x)

# second layer

x = Conv2D(filters = n\_filters, kernel\_size = (kernel\_size, kernel\_size),\

kernel\_initializer = 'he\_normal', padding = 'same')(input\_tensor)

if batchnorm:

x = BatchNormalization()(x)

x = Activation('relu')(x)

return x

**#standard u-net arch implementation**

def get\_unet(input\_img, n\_filters = 16, dropout = 0.1, batchnorm = True):

"""Function to define the UNET Model"""

# Contracting Path

c1 = conv2d\_block(input\_img, n\_filters \* 1, kernel\_size = 3, batchnorm = batchnorm)

p1 = MaxPooling2D((2, 2))(c1)

p1 = Dropout(dropout)(p1)

c2 = conv2d\_block(p1, n\_filters \* 2, kernel\_size = 3, batchnorm = batchnorm)

p2 = MaxPooling2D((2, 2))(c2)

p2 = Dropout(dropout)(p2)

c3 = conv2d\_block(p2, n\_filters \* 4, kernel\_size = 3, batchnorm = batchnorm)

p3 = MaxPooling2D((2, 2))(c3)

p3 = Dropout(dropout)(p3)

c4 = conv2d\_block(p3, n\_filters \* 8, kernel\_size = 3, batchnorm = batchnorm)

p4 = MaxPooling2D((2, 2))(c4)

p4 = Dropout(dropout)(p4)

c5 = conv2d\_block(p4, n\_filters = n\_filters \* 16, kernel\_size = 3, batchnorm = batchnorm)

**# Expansive Path**

u6 = Conv2DTranspose(n\_filters \* 8, (3, 3), strides = (2, 2), padding = 'same')(c5)

u6 = concatenate([u6, c4])

u6 = Dropout(dropout)(u6)

c6 = conv2d\_block(u6, n\_filters \* 8, kernel\_size = 3, batchnorm = batchnorm)

u7 = Conv2DTranspose(n\_filters \* 4, (3, 3), strides = (2, 2), padding = 'same')(c6)

u7 = concatenate([u7, c3])

u7 = Dropout(dropout)(u7)

c7 = conv2d\_block(u7, n\_filters \* 4, kernel\_size = 3, batchnorm = batchnorm)

u8 = Conv2DTranspose(n\_filters \* 2, (3, 3), strides = (2, 2), padding = 'same')(c7)

u8 = concatenate([u8, c2])

u8 = Dropout(dropout)(u8)

c8 = conv2d\_block(u8, n\_filters \* 2, kernel\_size = 3, batchnorm = batchnorm)

u9 = Conv2DTranspose(n\_filters \* 1, (3, 3), strides = (2, 2), padding = 'same')(c8)

u9 = concatenate([u9, c1])

u9 = Dropout(dropout)(u9)

c9 = conv2d\_block(u9, n\_filters \* 1, kernel\_size = 3, batchnorm = batchnorm)

outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)

model = Model(inputs=[input\_img], outputs=[outputs])

return model

**#input image to unet fn with adam optimizer and loss fn as binary crossentropy**

input\_img = Input((im\_height, im\_width, 1), name='img')

model = get\_unet(input\_img, n\_filters=16, dropout=0.05, batchnorm=True)

model.compile(optimizer=Adam(), loss="binary\_crossentropy", metrics=["accuracy"])

**#displaying the summary of model**

model.summary()

**#saving only the best model while training the images data**

**#learning rate lr set to 0.00001.**

callbacks = [

EarlyStopping(patience=10, verbose=1),

ReduceLROnPlateau(factor=0.1, patience=5, min\_lr=0.00001, verbose=1),

ModelCheckpoint('model-tgs-salt.h5', verbose=1, save\_best\_only=True, save\_weights\_only=True)

]

**#traing the model for 3600 images in 50 epochs to save only the best model.**

results = model.fit(X\_train, y\_train, batch\_size=32, epochs=30, callbacks=callbacks,\

validation\_data=(X\_valid, y\_valid))

**#ploting the model accuracy**

plt.figure(figsize=(8, 8))

plt.title("Learning curve")

plt.plot(results.history["loss"], label="loss")

plt.plot(results.history["val\_loss"], label="val\_loss")

plt.plot( np.argmin(results.history["val\_loss"]), np.min(results.history["val\_loss"]), marker="x", color="r", label="best model")

plt.xlabel("Epochs")

plt.ylabel("log\_loss")

plt.legend();

**# load the best model**

model.load\_weights('model-tgs-salt.h5')

**# Evaluate on validation set (this must be equals to the best log\_loss)**

model.evaluate(X\_valid, y\_valid, verbose=1)

**# Predict on train, val and test**

preds\_train = model.predict(X\_train, verbose=1)

preds\_val = model.predict(X\_valid, verbose=1)

**# Threshold predictions**

preds\_train\_t = (preds\_train > 0.5).astype(np.uint8)

preds\_val\_t = (preds\_val > 0.5).astype(np.uint8)

**#ploting the required images and their respective predictions**

def plot\_sample(X, y, preds, binary\_preds, ix=None):

"""Function to plot the results"""

if ix is None:

ix = random.randint(0, len(X))

has\_mask = y[ix].max() > 0

fig, ax = plt.subplots(1, 4, figsize=(20, 10))

ax[0].imshow(X[ix, ..., 0], cmap='seismic')

if has\_mask:

ax[0].contour(y[ix].squeeze(), colors='k', levels=[0.5])

ax[0].set\_title('Seismic')

ax[1].imshow(y[ix].squeeze())

ax[1].set\_title('Salt')

ax[2].imshow(preds[ix].squeeze(), vmin=0, vmax=1)

if has\_mask:

ax[2].contour(y[ix].squeeze(), colors='k', levels=[0.5])

ax[2].set\_title('Salt Predicted')

ax[3].imshow(binary\_preds[ix].squeeze(), vmin=0, vmax=1)

if has\_mask:

ax[3].contour(y[ix].squeeze(), colors='k', levels=[0.5])

ax[3].set\_title('Salt Predicted binary');

#calling the plot function. chnage the image\_lable to predict multiple images

# Check if training data looks all right

plot\_sample(X\_train, y\_train, preds\_train, preds\_train\_t, ix=14)

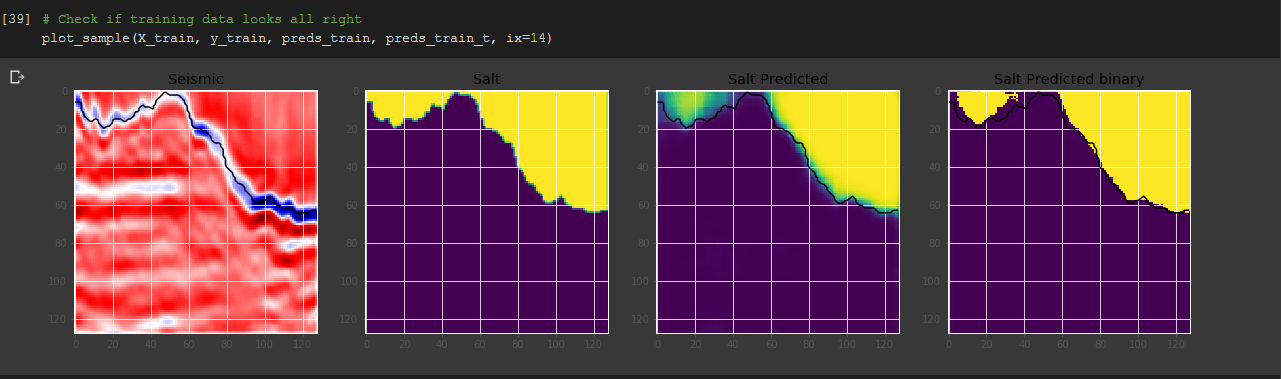
**# Check if valid data looks all right**

plot\_sample(X\_valid, y\_valid, preds\_val, preds\_val\_t, ix=19)

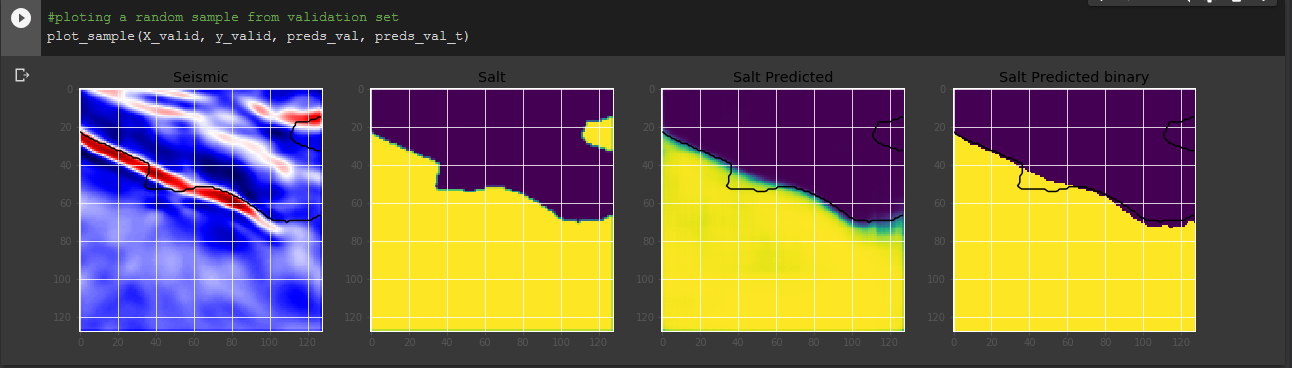
**APPENDIX-2:**

**OUTPUT AND RESULTS:**

**TRAINING RESULT:**



**TEST RESULT:**



**REFRENCES:**

[1] Twitter Sentiment Analysis using combined LSTM-CNN Models

Pedro M. Sosa -June 7, 2017

[2] EFFECTIVENESS OF U-NET IN DE-NOISING RGB IMAGES, by Rina Komatsu and Tad Gonsalves

Department of Information & Communication Sciences, Faculty of Science & Technology, Sophia University, Tokyo, Japan – August 2019

[3] The Effectiveness of Data Augmentation in Image Classification using Deep Learning

Jason Wang - Stanford University 450 Serra Mall zwang01@stanford.edu Luis Perez Google - 1600 Amphitheatre Parkway

[4] REVIEW ON IMAGE SEGMENTATION TECHNIQUES

1.Associate Professor Dr.S.Kannan

Department of Computer Applications, Madurai Kamaraj University.

2. Research Scholar, Vairaprakash Gurusamy,

Department of Computer Applications, Madurai Kamaraj University.

[5] Fully Convolutional Networks for Semantic Segmentation

Jonathan Long, Evan Shelhamer, Trevor Darrell.

UC Berkeley {jonlong,shelhamer,trevor}@cs.berkeley.edu.