National College of Ireland

Configuration Manual

Msc Research Project
Data Analytics

Nikhil Chambial
Student ID: x20232713

School of Computing
National College of Ireland

Supervisor: Mr. Taimur Hafeez

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Nikhil Chambial
Student ID:	x20232713
Programme:	Data Analytics
Year:	2022
Module:	Msc Research Project
Supervisor:	Mr. Taimur Hafeez
Submission Due Date:	19/09/2022
Project Title:	**Configuration Manual**
Word Count:	1300
Page Count:	13

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	18th September 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Nikhil Chambial
x20232713
Msc Research Project

1 Introduction

This document is a configuration manual that specifies the various components of the research named - **"Deep Neural Network for Seismic Image Segmentation and Detection of Salt Domes"**. In addition, the document discusses the software and hardware requirements for executing this project, along with the model design, implementation details, and evaluation of the model.

2 System Configuration

2.1 Hardware

- **Processor:** Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
- **RAM:** 16GB
- **Operating System:** 64-bit Windows OS, x64-based processor
- **GPU:** Intel(R) UHD Graphics 620, 6GB
- **Storage:** 1 TB

2.2 Software

- **Google Collaboratory:** Google Collaboratory [1] is a free cloud-based platform provided by Google for machine learning development. The platform provides similar functionality to Jupyter Notebook and optional GPU and TPU hardware accelerator to speed up the code execution. In addition, the GPU hardware accelerator is very efficient in handling large datasets.

[1] https://colab.research.google.com/

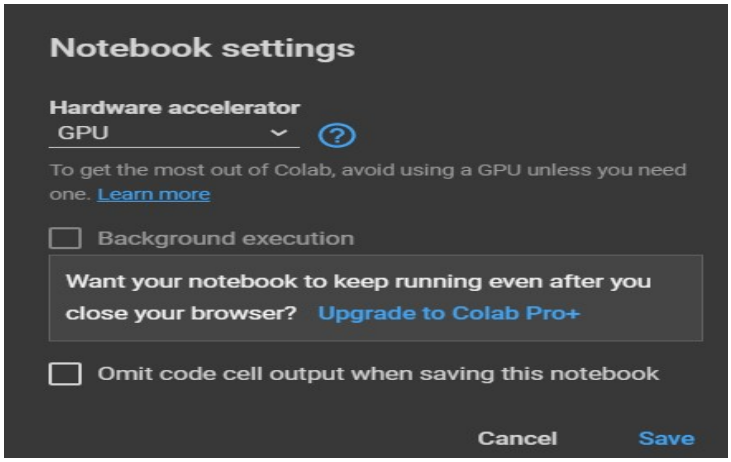

Figure 1: Google Collab: Hardware Accelerator

- **Microsoft Excel:** Software by Microsoft to create visualizations and tables used in the report.
- **Overleaf Latex:** Overleaf platform [2] was used to create project report and configuration manual using Latex.

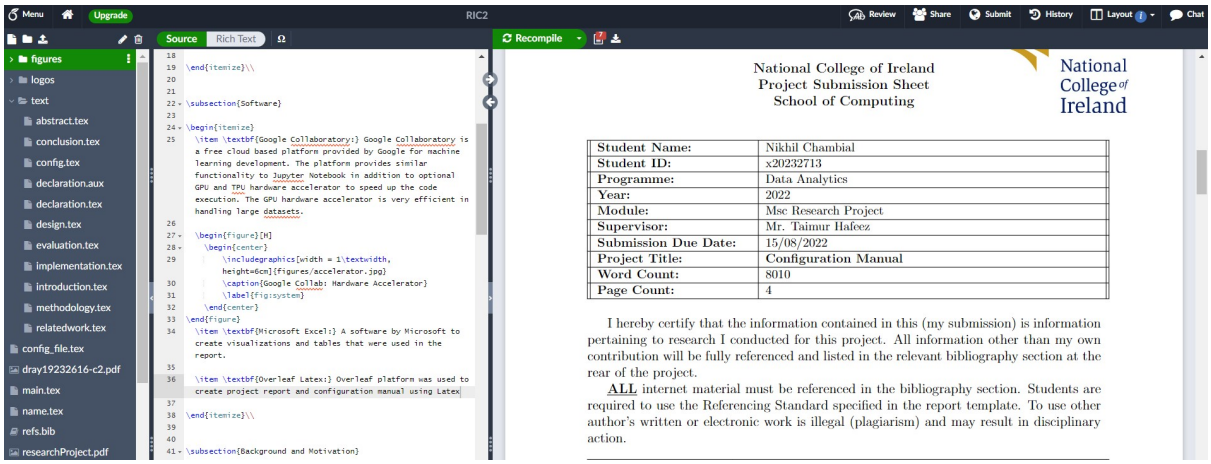

Figure 2: Overleaf Platform

3 Project Development

The project was implemented using the python programming language. The project can broadly be divided into three stages of development- Data Preparation, Model Implementation, and Evaluation of the Model.

[2]https://www.overleaf.com/

3.1 Data Extraction and Staging

The dataset in this project was made available by a Geophysical company called TGS through a competition hosted on Kaggle [3], a public platform.

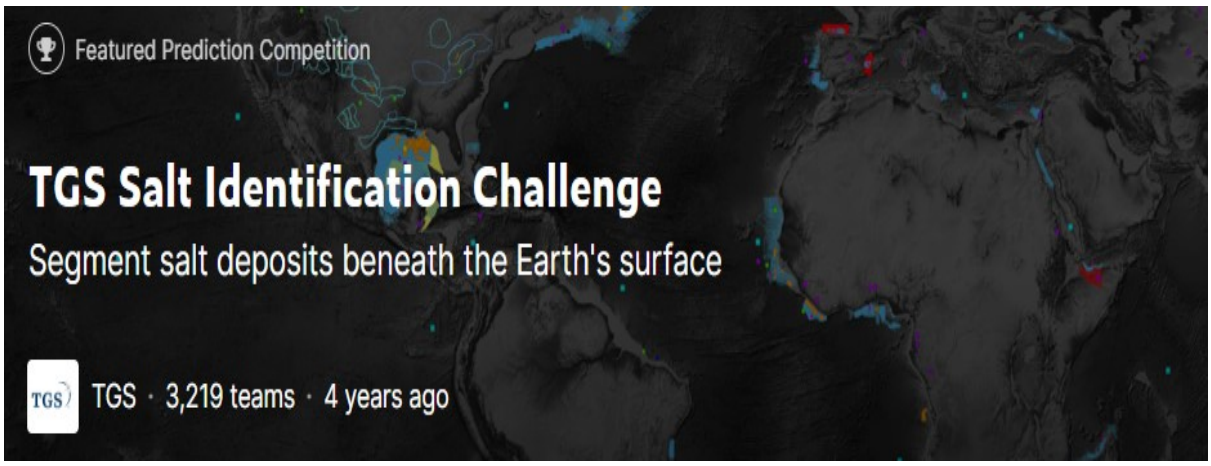

Figure 3: Overleaf Platform

The dataset was first downloaded as a zip file from Kaggle and uploaded on a personal google drive. It can be accessed using the below link:
TGS Salt Dataset

Below are the steps to use this dataset:

- **Step 1:** Download this dataset and upload to your Google Drive.
- **Step 2:** Now login to Google Collab and run the code file named: **20232713_Project_Code**
- **Step 3:** In the second code cell, while running the code in Figure 4, the system will ask for the permission to use the same google drive. Once given permission, the code will stage the stage to notebook and unzip it automatically.

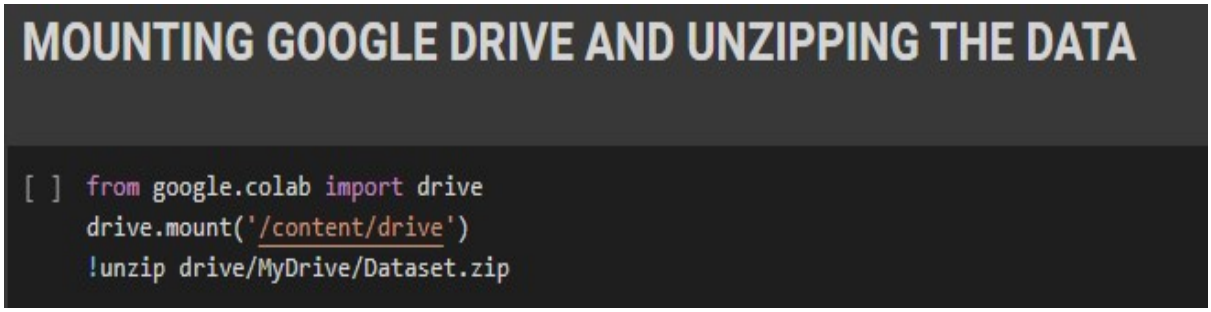

Figure 4: Google Drive Mount

[3] https://www.kaggle.com/competitions/tgs-salt-identification-challenge/data

3.2 Data Preparation

The seismic image data was imported using Pandas data frames and Numpy arrays. The data pre-processing was carried out following the below steps:

3.2.1 Data Loading

The following figures show the code snippets for various data extraction and manipulation activities. Figure 5 shows the various libraries that were imported as part of the development.

IMPORTING LIBRARIES

```
[ ]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     import os, sys, random, warnings, math
     import cv2
     from skimage.io import imread, imshow, concatenate_images
     from skimage.transform import resize
     from skimage.morphology import label
     from tensorflow.keras.preprocessing.image import array_to_img, img_to_array, load_img, ImageDataGenerator
     import tensorflow as tf
     from tensorflow.keras import backend as K
     from tensorflow.keras import models, Input, layers, callbacks, utils, optimizers
     from tqdm.auto import tqdm, trange
     from itertools import chain
```

Figure 5: Importing Libraries

After the libraries were imported, the Google Drive was mounted, and the data was unzipped. As seen in the Figure 6, the dataset folder contains Test Image and Train Image sub-folders along with CSV files containing depth and training data information.

MOUNTING GOOGLE DRIVE AND UNZIPPING THE DATA

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
     !unzip drive/MyDrive/Dataset.zip

     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
     Archive:  drive/MyDrive/Dataset.zip
     replace Dataset/depths.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename:

[ ]  ls/content/Dataset/

     depths.csv  sample_submission.csv  test/  train/  train.csv
```

Figure 6: Importing Libraries

3.2.2 Data Visualization

Figure 7 shows the code implemented to visualize the seismic image. The figure shows seismic images along with the respective salt masks.

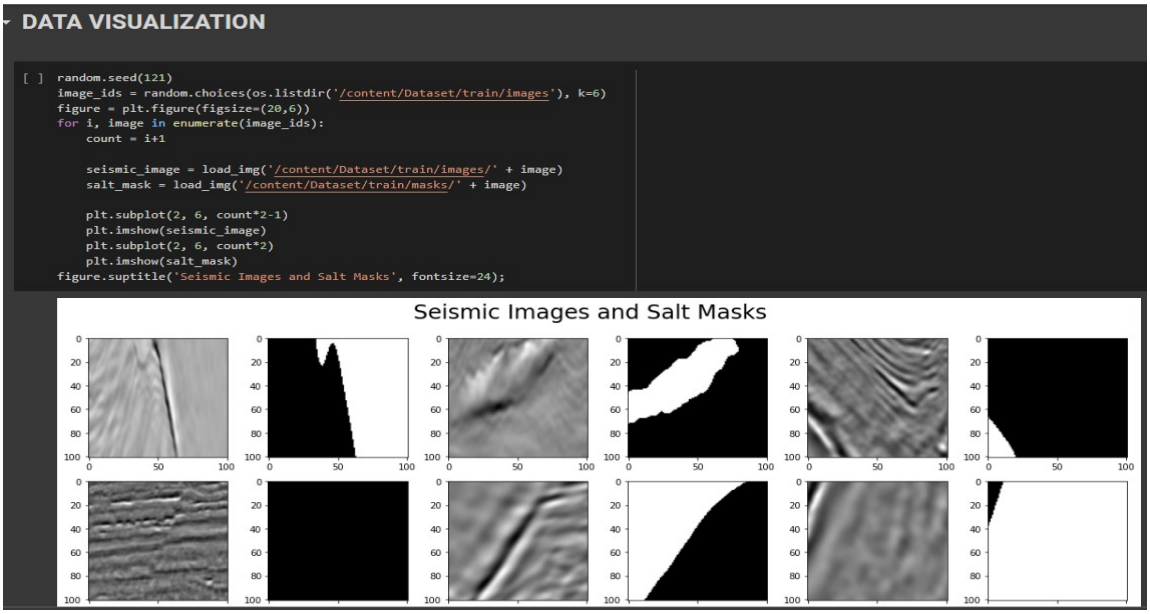

Figure 7: Seismic Image and Salt Mask

3.2.3 Data Manipulation

All the seismic images and respective salt masks were loaded in grayscale format and resized to 128*128 dimensions from 101*101 using resize() method. The images were also converted to the array using img_to_array method. There are 4000 images and 4000 salt masks, as seen in Figure 8.

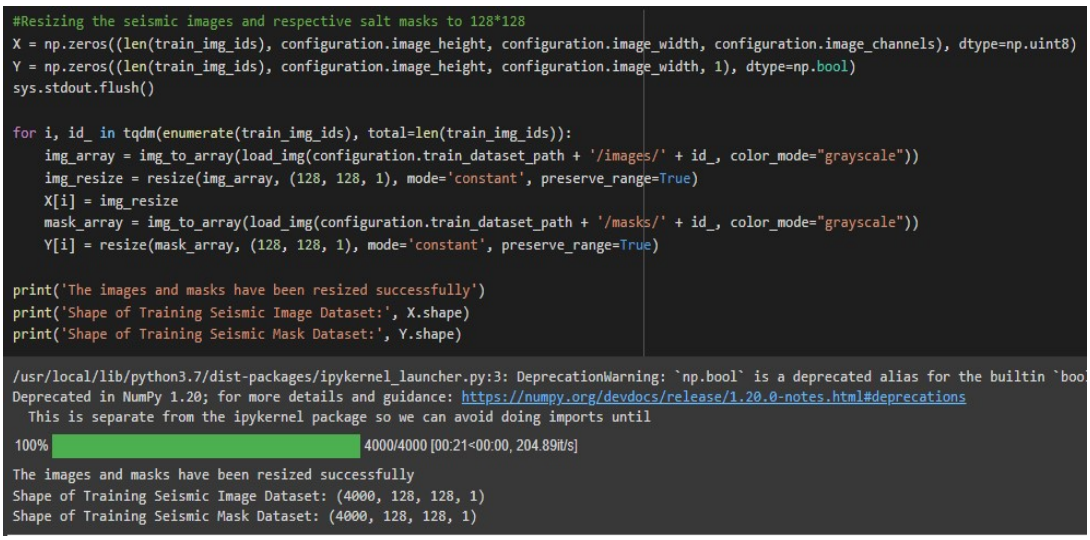

Figure 8: Data Manipulation

3.2.4 Data Augmentation and Split

Finally, the data was augmented using various data augmentation techniques such as flipping left-right and up-down using the fliplr() and flipus() methods to increase the dataset to 12000 images and 12000 salt masks, as seen in Figure 9.

DATA SPLITTING

```
[ ]  # Dataset Split into Training and Validation Datatset
     X_train_data = X[:int(0.8*len(X))]
     Y_train_data = Y[:int(0.8*len(X))]
     X_eval_data  = X[int(0.8*len(X)):]
     Y_eval_data  = Y[int(0.8*len(X)):]
```

DATA AUGMENTATION

```
[ ]  # Data Augmentation by flipping the Images and Masks up-down and left-right
     X_train_data = np.append(X_train_data, [np.fliplr(x) for x in X], axis=0)
     Y_train_data = np.append(Y_train_data, [np.fliplr(x) for x in Y], axis=0)
     X_train_data = np.append(X_train_data, [np.flipud(x) for x in X], axis=0)
     Y_train_data = np.append(Y_train_data, [np.flipud(x) for x in Y], axis=0)

     del X, Y

     print('Seismic Image Training Dataset shape:', X_train_data.shape)
     print('Seismic Image Validation Dataset shape:', X_eval_data.shape)
     print('Seismic Image Training Dataset shape:', Y_train_data.shape)
     print('Seismic Image Validation Dataset shape:', Y_eval_data.shape)

     Seismic Image Training Dataset shape: (11200, 128, 128, 1)
     Seismic Image Validation Dataset shape: (800, 128, 128, 1)
     Seismic Image Training Dataset shape: (11200, 128, 128, 1)
     Seismic Image Validation Dataset shape: (800, 128, 128, 1)
```

Figure 9: Data Augmentation

3.3 Modelling

3.3.1 Model Building Blocks

Three functions were defined to form building blocks of the U-net deep learning model. BatchActivate() function is used to first perform batch normalization on the input layer and then activate using the relu activation layer. The convolution_block() function takes the input layer, filers size, and stride size as input and applies Conv2D() and BatchActivate operation on the input. The residual_block() function first activates the input and then applies two convolution operations on the input, as seen in Figure 10.

```
#Defining different functions like Batch activate , convolution block and residual block
#These functions are used as building blocks for the Unet architecture
def BatchActivate(x):
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    return x

def convolution_block(x, filters, size, strides=(1,1), padding='same', activation=True):
    x = layers.Conv2D(filters, size, strides=strides, padding=padding)(x)
    if activation == True:
        x = BatchActivate(x)
    return x

def residual_block(blockInput, num_filters=16, batch_activate = False):
    x = BatchActivate(blockInput)
    x = convolution_block(x, num_filters, (3,3) )
    x = convolution_block(x, num_filters, (3,3), activation=False)
    x = layers.Add()([x, blockInput])
    if batch_activate:
        x = BatchActivate(x)
    return x
```

Figure 10: Unet Building Blocks

3.3.2 Unet Model Implementation

The Unet architecture (Ronneberger et al.; 2015) was implemented using the above model building blocks with four levels in the encoder part, 1 in the middle, and four in the decoder part. Each level in the encoder consisted of a Conv2D() layer, two residual blocks, a MaxPooling2D() layer, and a Dropout() layer. In the decoder part, Conv2D() was replaced by Conv2DTranspose() operation, and each layer was concatenated with the output of encoder layers before the Dropout() operation. The DropoutRatio was initiated as 0.5 and was halved at every level in the encoder Dropout() operation.

```
def my_unet_model(input_layer, start_neurons, DropoutRatio = 0.5):
    scaled = layers.Lambda(lambda x: x / 255) (input_layer)

    # Encoder or Contraction Part of Unet
    c1 = layers.Conv2D(start_neurons * 1, (3, 3), activation=None, padding="same")(scaled)
    c1 = residual_block(c1,start_neurons * 1)
    c1 = residual_block(c1,start_neurons * 1, True)
    p1 = layers.MaxPooling2D((2, 2))(c1)
    p1 = layers.Dropout(DropoutRatio/2)(p1)


    c2 = layers.Conv2D(start_neurons * 2, (3, 3), activation=None, padding="same")(p1)
    c2 = residual_block(c2,start_neurons * 2)
    c2 = residual_block(c2,start_neurons * 2, True)
    p2 = layers.MaxPooling2D((2, 2))(c2)
    p2 = layers.Dropout(DropoutRatio)(p2)


    c3 = layers.Conv2D(start_neurons * 4, (3, 3), activation=None, padding="same")(p2)
    c3 = residual_block(c3,start_neurons * 4)
    c3 = residual_block(c3,start_neurons * 4, True)
    p3 = layers.MaxPooling2D((2, 2))(c3)
    p3 = layers.Dropout(DropoutRatio)(p3)


    c4 = layers.Conv2D(start_neurons * 8, (3, 3), activation=None, padding="same")(p3)
    c4 = residual_block(c4,start_neurons * 8)
    c4 = residual_block(c4,start_neurons * 8, True)
    p4 = layers.MaxPooling2D((2, 2))(c4)
    p4 = layers.Dropout(DropoutRatio)(p4)

    # Middle of Unet Architecture
    c5 = layers.Conv2D(start_neurons * 16, (3, 3), activation=None, padding="same")(p4)
```

Figure 11: Unet Implementation

3.3.3 Model Parameters

Before training the model, two functions were created to improve the model training. The iou_vector() function was created to use as the evaluation metric while training the model. The model was trained using a binary cross entropy loss function, but another loss function called weighted_cross_entropy() was also implemented to compare the performance.

```python
#Defining Unet model evaluation metric IOU(Intersection over Union)
def iou_vector(R, S):
    batch_size = R.shape[0]
    metric = []
    for x in range(batch_size):
        t, p = R[x]>0, S[x]>0
        intersection = np.logical_and(t, p)
        union = np.logical_or(t, p)
        iou = (np.sum(intersection > 0) + 1e-10 )/ (np.sum(union > 0) + 1e-10)
        thresholds = np.arange(0.5, 1, 0.05)
        k = []
        for y in thresholds:
            k.append(iou > y)
        metric.append(np.mean(k))
    return np.mean(metric)

def iou_metric(label, pred):
    return tf.numpy_function(iou_vector, [label, pred>0.5], tf.float64)
```

Figure 12: iou_vector evaluation metric

```python
# Defining Weighted Cross Entropy Loss Function to compare the performnace with Biinary Cross Entropy
def weighted_cross_entropy(i):
  def loss(y_true, y_pred):
    weight1 = i * tf.cast(y_true, tf.float32)
    weight2 = 1 - tf.cast(y_true, tf.float32)

    z = (tf.math.log1p(tf.exp(-tf.abs(y_pred))) + tf.nn.relu(-y_pred)) * (weight1 + weight2) + y_pred * weight2
    return tf.reduce_mean(z)

  return loss
```

Figure 13: weighted_cross_entropy() loss function

3.3.4 Model Building

The model was compiled using **adam** optimizer and **binary_crossentropy** loss function. The metrics used while compiling the model were **accuracy** and **iou_metric** discussed above. Different combinations of optimizers and loss functions were also implemented, such as Nadam, SGD, Adamax optimizer, and weighted_cross_entropy() loss function.

```
unet = models.Model(unet_input_layer, unet_output_layer)
unet.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc',iou_metric])
#Use loss=weighted_cross_entropy(beta=0.9) for Weighted Cross Entropy
#Use optimizers = nadam, adamax, sgd to compare results
unet.summary()
```

Figure 14: Model Compile

3.4 Evaluation

3.4.1 Model Training Hyper-parameters

The implementation of Keras callbacks was done before training the model. A value of 30 was set for the Early Stopping parameter, and the learning rate decayed to 0.1 when the accuracy did not improve for five epochs. Min value of the learning rate is set to 1e-12. By using these parameters, the model was run over 200 epochs with a batch size of 32.

```
MODEL TRAINING

[ ] early_stopping = callbacks.EarlyStopping(patience=30, verbose=1, restore_best_weights=True)
    reduce_on_plateau = callbacks.ReduceLROnPlateau(factor=0.1, patience=5, min_lr=1e-12, verbose=1)

    results = unet.fit(
        X_train, Y_train, validation_data=(X_eval, Y_eval), batch_size=32, epochs=4, callbacks=[early_stopping, reduce_on_plateau]
    )
```

Figure 15: Model Training

The model training stopped at the 74th epoch due to early stopping parameters set before model training. As seen in Figure 15, the model was able to achieve an accuracy of around 0.97 and iou_metric of 0.79.

```
Epoch 68: ReduceLROnPlateau reducing learning rate to 1.0000000082740371e-09.
350/350 [==============================] - 75s 215ms/step - loss: 0.0861 - acc: 0.9670 - iou_metric: 0.7749 - val_loss: 0.0953 - val_acc: 0.9635 - val_iou_metric: 0.7940 - lr: 1.0000e-08
Epoch 69/200
350/350 [==============================] - 75s 215ms/step - loss: 0.0867 - acc: 0.9669 - iou_metric: 0.7729 - val_loss: 0.0954 - val_acc: 0.9636 - val_iou_metric: 0.7956 - lr: 1.0000e-09
Epoch 70/200
350/350 [==============================] - 76s 218ms/step - loss: 0.0855 - acc: 0.9671 - iou_metric: 0.7739 - val_loss: 0.0953 - val_acc: 0.9636 - val_iou_metric: 0.7935 - lr: 1.0000e-09
Epoch 71/200
350/350 [==============================] - 76s 218ms/step - loss: 0.0852 - acc: 0.9675 - iou_metric: 0.7732 - val_loss: 0.0953 - val_acc: 0.9636 - val_iou_metric: 0.7959 - lr: 1.0000e-09
Epoch 72/200
350/350 [==============================] - 75s 215ms/step - loss: 0.0853 - acc: 0.9673 - iou_metric: 0.7739 - val_loss: 0.0952 - val_acc: 0.9637 - val_iou_metric: 0.7947 - lr: 1.0000e-09
Epoch 73/200
350/350 [==============================] - ETA: 0s - loss: 0.0851 - acc: 0.9673 - iou_metric: 0.7736Restoring model weights from the end of the best epoch: 43.

Epoch 73: ReduceLROnPlateau reducing learning rate to 1.0000000082740371e-10.
350/350 [==============================] - 76s 216ms/step - loss: 0.0851 - acc: 0.9673 - iou_metric: 0.7736 - val_loss: 0.0953 - val_acc: 0.9636 - val_iou_metric: 0.7959 - lr: 1.0000e-09
Epoch 73: early stopping
```

Figure 16: Training Output

Figure 16 shows the accuracy and loss function plots of the Unet model. The graphs were plotted using the history() function of the model output and the subplots() function to draw the graphs.

```
a, (fig1, fig2, fig3) = plt.subplots(1, 3, figsize=(30, 6))
t = a.suptitle('Unet Performance', fontsize=12)
a.subplots_adjust(top=0.85, wspace=0.3)
epoch_list = results.epoch

fig1.plot(epoch_list, results.history['acc'], label='Train Accuracy')
fig1.plot(epoch_list, results.history['val_acc'], label='Validation Accuracy')
fig1.set_xticks(np.arange(0, epoch_list[-1], 5))
fig1.set_ylabel('Accuracy Value');fig1.set_xlabel('Epoch');fig1.set_title('Accuracy')
fig1.legend(loc="best");fig1.grid(color='gray', linestyle='-', linewidth=0.5)

fig2.plot(epoch_list, results.history['loss'], label='Train Loss')
fig2.plot(epoch_list, results.history['val_loss'], label='Validation Loss')
fig2.set_xticks(np.arange(0, epoch_list[-1], 5))
fig2.set_ylabel('Loss Value');fig2.set_xlabel('Epoch');fig2.set_title('Loss')
fig2.legend(loc="best");fig2.grid(color='gray', linestyle='-', linewidth=0.5)

fig3.plot(epoch_list, results.history['iou_metric'], label='Train IOU metric')
fig3.plot(epoch_list, results.history['val_iou_metric'], label='Validation IOU metric')
fig3.set_xticks(np.arange(0, epoch_list[-1], 5))
fig3.set_ylabel('IOU metric');fig3.set_xlabel('Epoch');fig3.set_title('IOU metric')
fig3.legend(loc="best");fig3.grid(color='gray', linestyle='-', linewidth=0.5)
```

Figure 17: Plotting Evaluation Plots

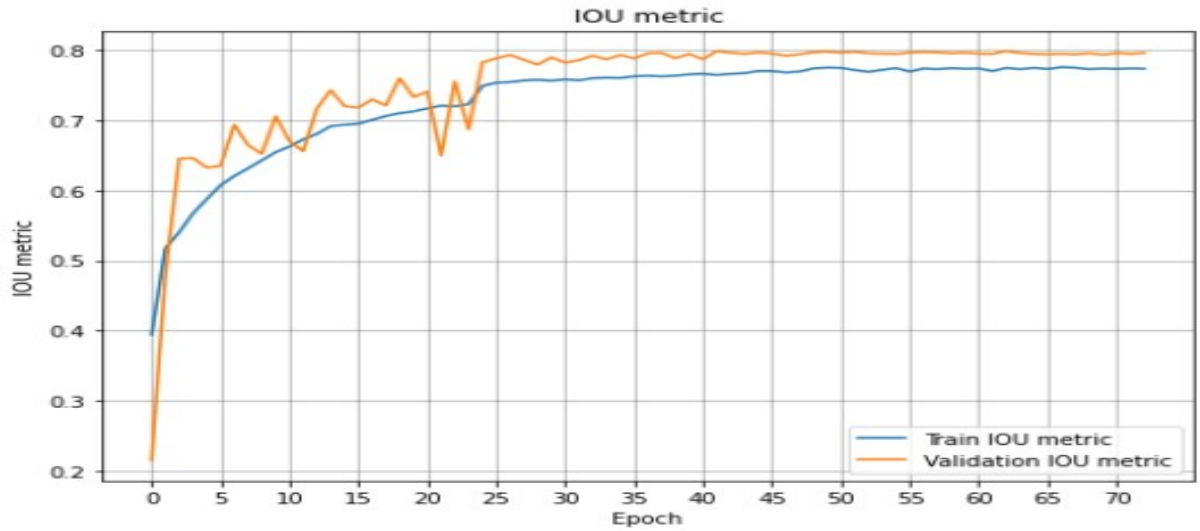

Figure 18: IoU Plot

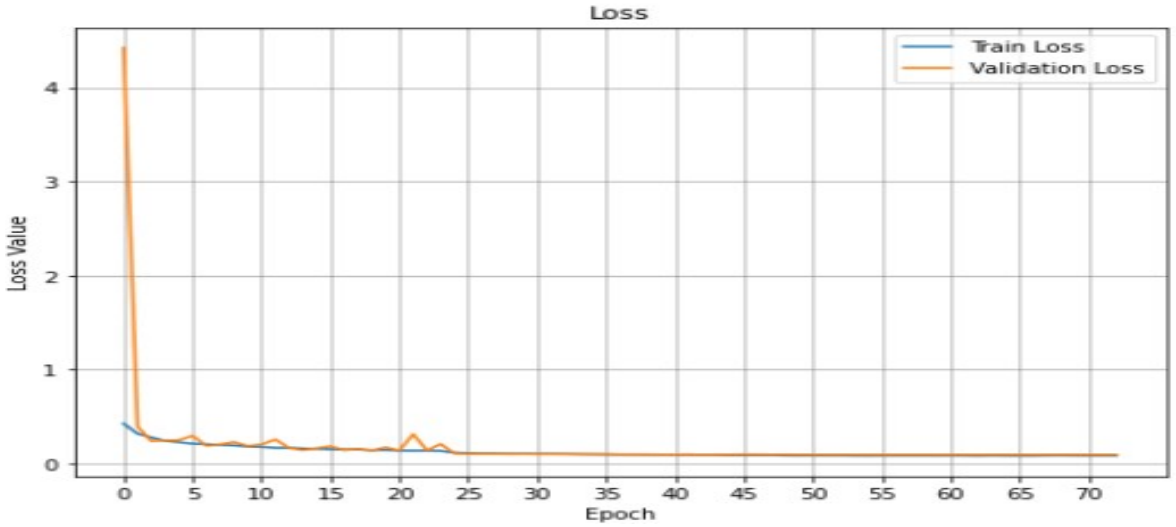

Figure 19: Loss vs Epochs

Finally, the output salt mask predicted by the model was compared with the ground truth salt mask provided in the dataset. As seen in the Figure 20, the model performs well at correctly detecting the salt region in the seismic image.

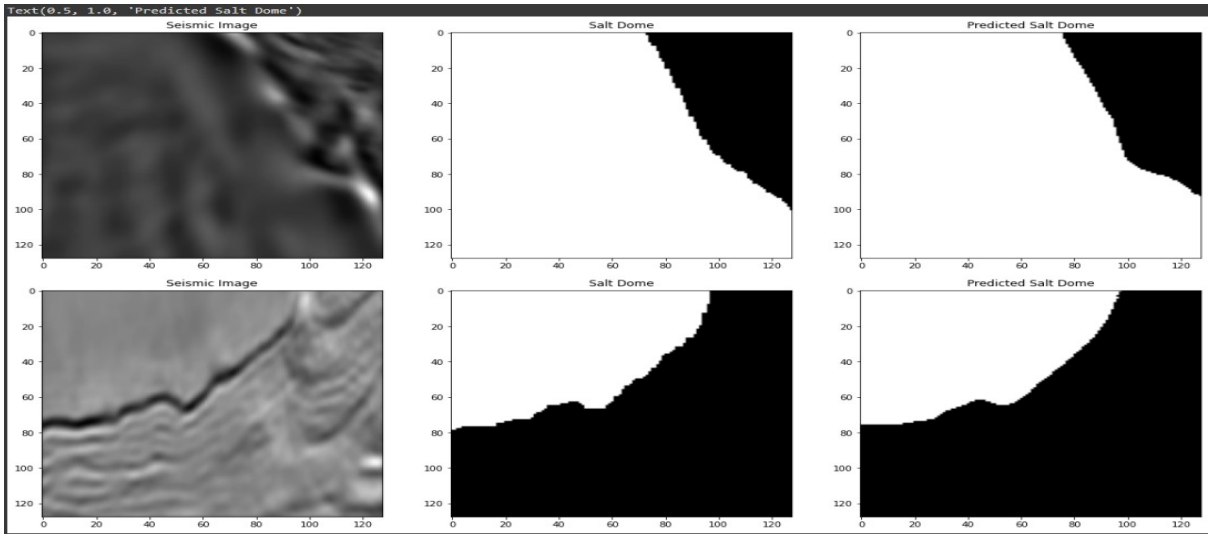

Figure 20: Ground Truth Salt Mask vs Predicted Salt Mask

References

Ronneberger, O., Fischer, P. and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation, *International Conference on Medical image computing and computer-assisted intervention*, Springer, pp. 234–241.