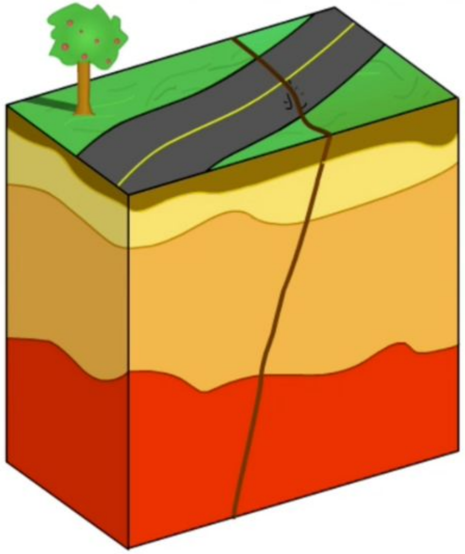
Information Systems Project

Fracture Detection From Seismic Images

~ Team 28



-Mohammed Fayiz Parappan

-Sneha Kumari

-Ronit Wanare

Table of Contents

Project Overview

Project Title: Fracture Detection from Seismic images

Problem Statement:

- Fracture/Fault detection from seismic images is an integral task both for geoscience research and industries involved in oil extraction.

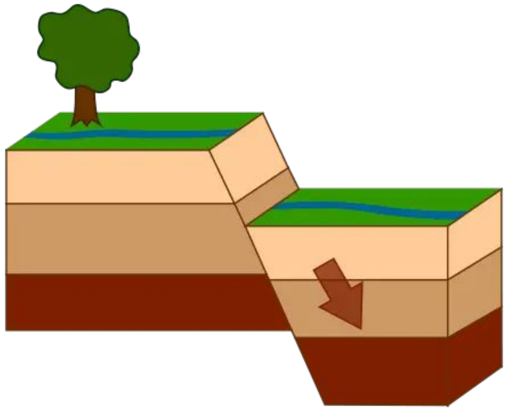
- Present detection technique involve tremendous task of printing inline seismic data on large sheets and raw observation across pixel columns on the sheet.

- Large number of workforce is required for faster detection.
- High Human error

- This results in significant cost of resources and time to identity faults. Present development in the field of deep learning and image segmentation have opened possibilities of segmenting fault layers.

System Description

A model to detect and present fault/fracture layer using deep learning techniques



Milestone/Deliverables

The Following are the milestone which we wished to achieve by the end of this project:-

- Performing Transfer learning on the prefitted U-net model(fault sec 3D) on synthetic seismic image dataset to extract features out of our model and to increase model accuracy.

- Designing and framing custom model as per user Input annotations, such as taking filter size, no.of Convolution layers and filter number as Inputs from the user.

- Designing and framing GUI supporting real-time access to user to operate fault sec 3D.

Once the **learning model** is integrated with the **UI** model and is being used successfully to predict **Faults** in seismic images

→

We will use this model to predict **Fractures** in seismic images and compare the results.

Software:-

-Qt Designer (for UI model)
-Notepad++(for ide purpose to edit code)

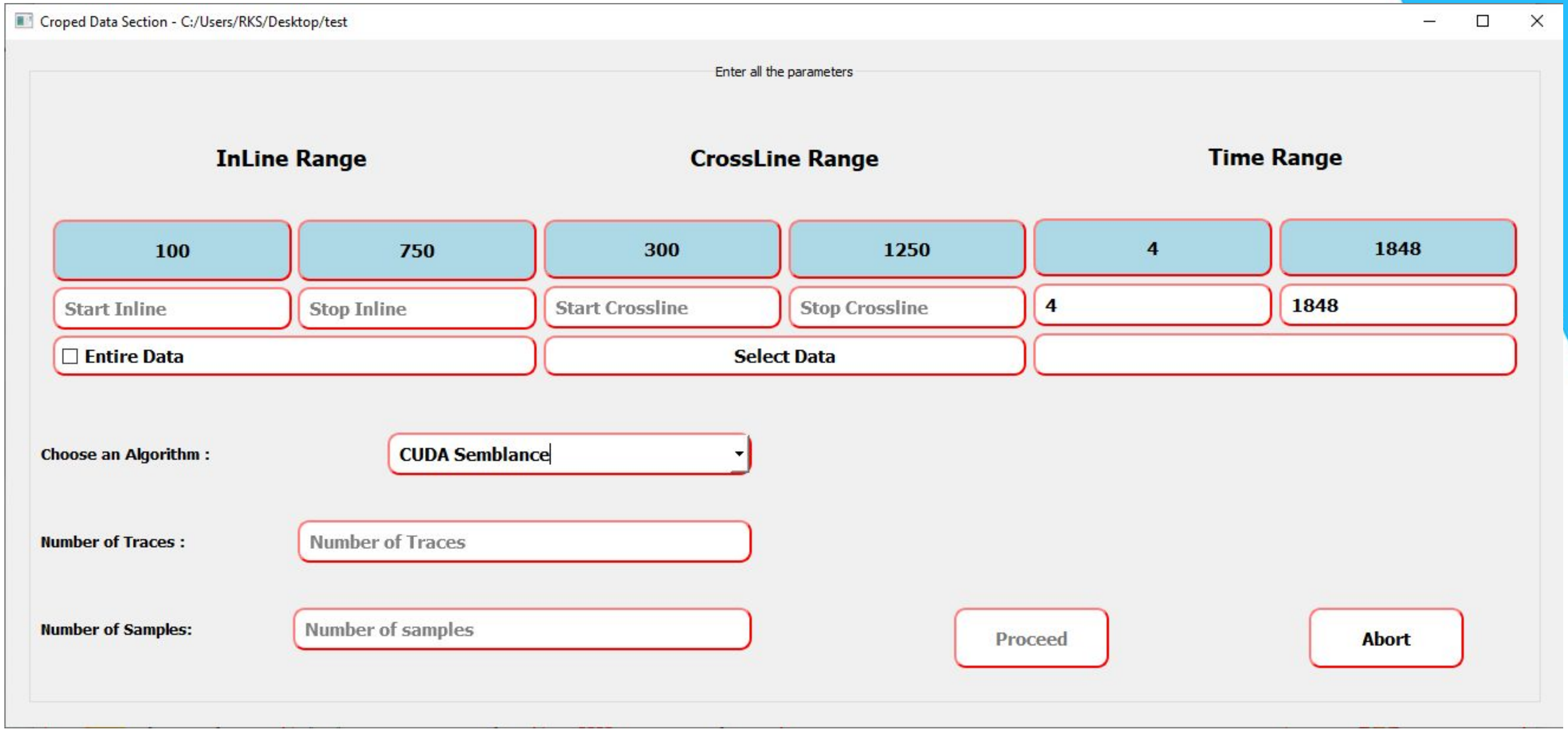
Operating System:-

-Windows

WindowsOS

Initial UI Design

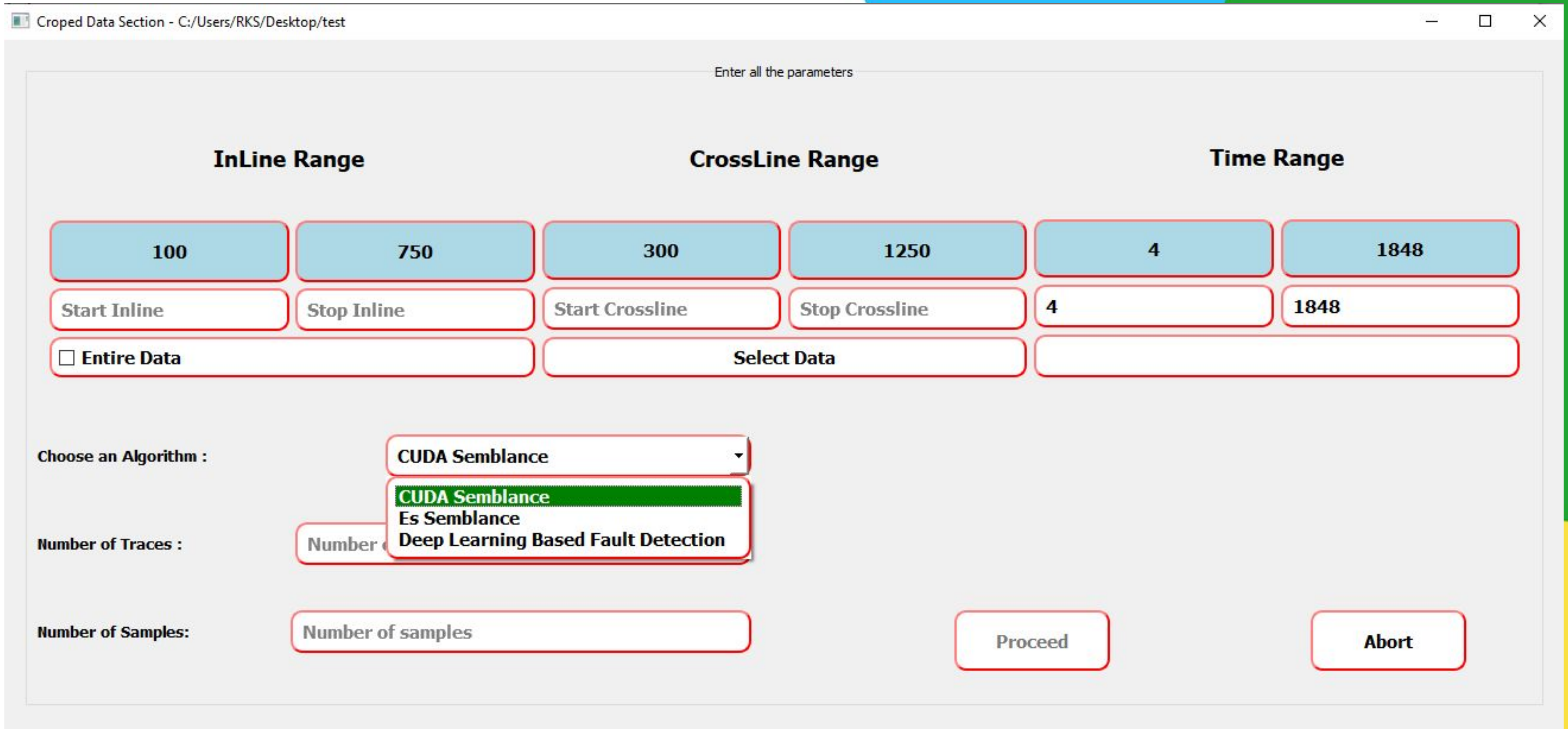
Page 1



Initial UI Design

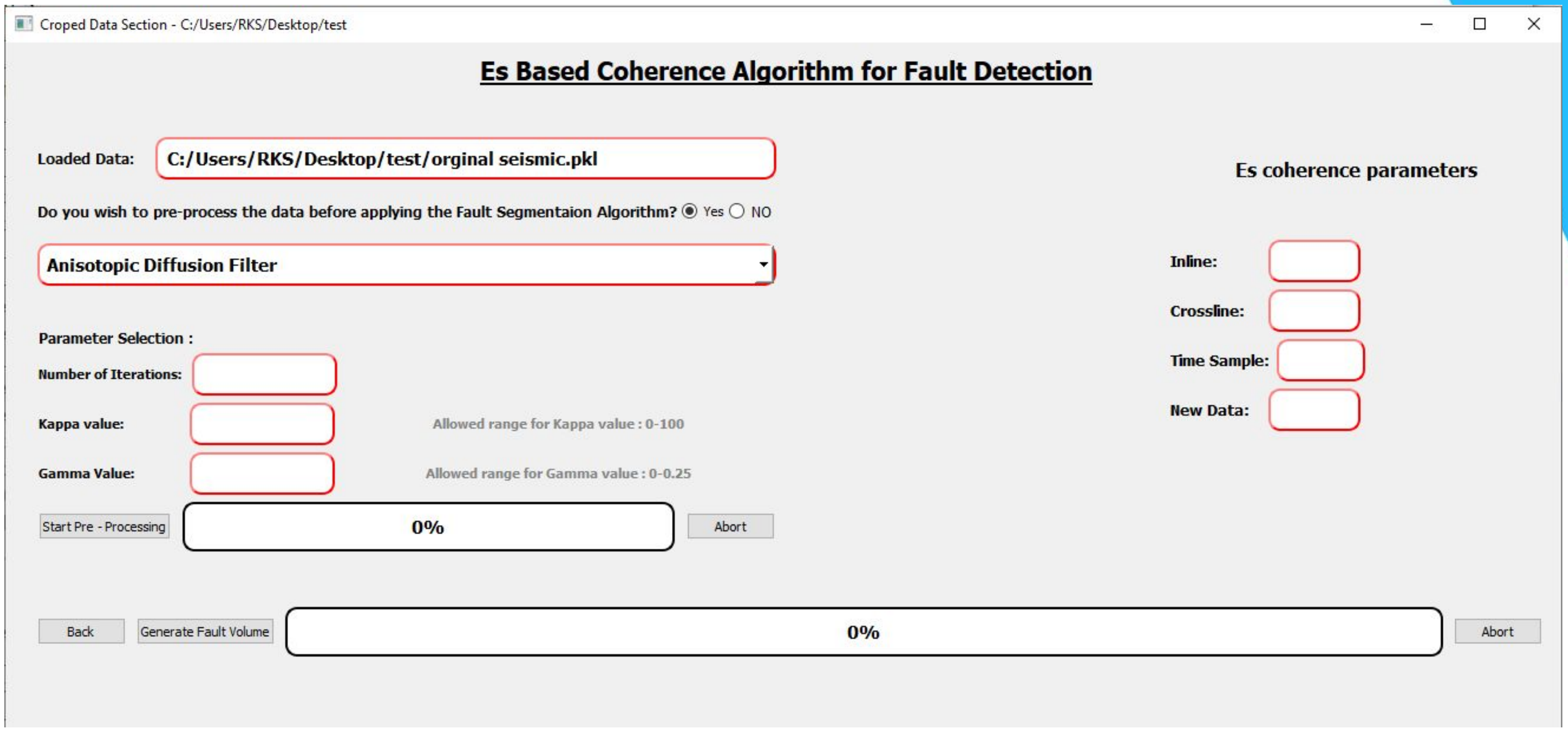
Page 1 -

Added new item in dropdown menu for Algorithm Selection and integrated it with the new page for Es based Coherence algorithm



Initial UI Design

Page 3 - Designed frontend and activated major key buttons and functions via backend scripting.



Live Demo

Remote control of system’s Lab

Code Snippet

Croped_DataUI.py - Conversion of UI model to Python file

This is the file that is generated from the Croped_Data.ui file conversion to python file by running the command:

```
>pysuic5 -x Croped_Data.ui -o Croped_DataUI.py
```

Croped_DataPy.py | Croped_DataUI.py × | EsCcode.py | CUDA_sem.py | MFP_main.py

C: > Users > sneha > Downloads > MFPGUI - updated > Croped_DataUI.py

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'Croped_Data.ui'
#
# Created by: PyQt5 UI code generator 5.9.2
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(1286, 605)
        Form.setCursor(QtGui.QCursor(QtCore.Qt.ArrowCursor))
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap("icons/ONGC.png"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
        Form.setWindowIcon(icon)
        self.verticalLayout_13 = QtWidgets.QVBoxLayout(Form)
        self.verticalLayout_13.setObjectName("verticalLayout_13")
        self.stackedWidget = QtWidgets.QStackedWidget(Form)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.MinimumExpanding, QtWidgets.QSizePolicy.Expanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.stackedWidget.sizePolicy().hasHeightForWidth())
        self.stackedWidget.setSizePolicy(sizePolicy)
```

Code Snippet

Croped_DataPy.py - Python file to activates clicks

This is the file used to define all key activations of buttons and clicks.

Croped_DataPy.py × | Croped_DataUI.py | EsCcode.py | CUDA_sem.py | MFP_m

C: > Users > sneha > Downloads > MFPGUI - updated > Croped_DataPy.py

```python
        self.radioButton_yes_3.toggled.connect(self.showAlgorithm_sem)
        self.radioButton_yes_28.toggled.connect(self.showAlgorithm_es)
        self.solver_available=[]
        self.solver_available.append('Dual_Simplex_IIT_KGP')
        self.solver_available.append('GUROBI')
        self.optimize_comboBox.addItems(list(self.solver_available))
        self.optimize_comboBox.setCurrentIndex(1)
        self.setWindowTitle('Croped Data Section - '+self.Base_directory)
        self.lineEdit_29.setText(self.Base_directory+"/orginal seismic.pkl")
        self.lineEdit_183.setText(self.Base_directory+"/orginal seismic.pkl")

    def showPreProcess_sem(self):
        if self.comboBox_3.currentIndex() == 0:
            self.frame_9.show()
        else:
            self.frame_9.hide()

    def showAlgorithm_sem(self):
        if self.radioButton_yes_3.isChecked():
            self.predictOnPreProcess=True
            self.comboBox_3.show()
        else:
```

Code Snippet

EsCode.py and CUDA_Sem.py

These are the main algorithm file.

EsCcode.py × | CUDA_sem.py | Croped_DataPy.py

C: > Users > sneha > Downloads > MFPGUI - updated > EsCcode.py

```python

    def run(self):
    #def compute_semblance(data_compute,w1,w2,w3
        try:
            self.inline_aug= int(np.floor(self.w
            self.xline_aug=int(np.floor(self.w2/
            self.samples_aug=int(self.w3)
            self.newdata_aug=int(self.w4)
            self.seis_vol=np.transpose(self.seis
            self.seis_vol = np.ascontiguousarray
            self.data_aug=np.zeros([self.seis_vc
            self.data_aug[ self.inline_aug:self.

```

CUDA_sem.py × | Croped_DataPy.py | Croped_Data

C: > Users > sneha > Downloads > MFPGUI - updated > CUDA_s

```python

def compute_semblance(data_compute,w1,w2,w3,p
    semblance_result=np.zeros(data_compute.sh
    inline_start=int(np.floor(w1/2))
    print(int(np.floor(w2/2)),int(np.floor(w3
    w2_gpu=int(np.floor(w2/2))
    w3_gpu=int(np.floor(w3))
    print(type(w3_gpu))
    progress_callback = progress if progress
    n=(1/(data_compute.shape[0]-2*inline_star
    for i in range(inline_start,data_compute.
        small_data=data_compute[i-inline_star
      # print(i)
        A_global_mem = cuda.to_device(small_c
```

Further Tasks to be implemented -

- EsCode.py - algorithm to be defined properly as run function()

- Bind EigenCode file similar to EsCode and CUIDA_Sem

- Pre-Process button to be activated in Es based coherence algorithm page.

Learning Algorithms

Feasibility Study

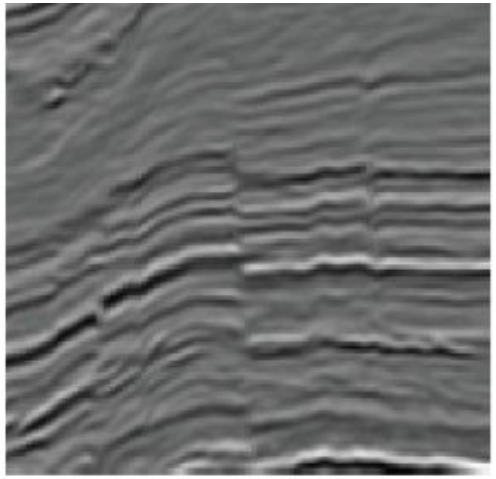
Objective:- Detect and Present Fractures on seismic images

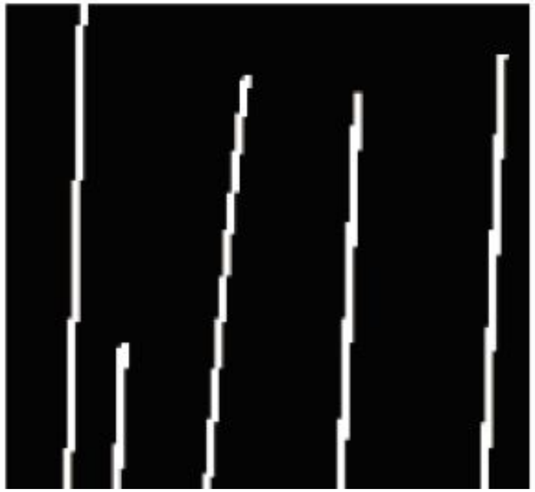
Methods Explored frequently in research papers:-

- Image segmentation using DL
- Transfer learning from a convolutional neural network pre-trained with synthetic seismic data
- Designing GUI to adjust and access the model as per user needs.

How Deep Learning Can Help ?

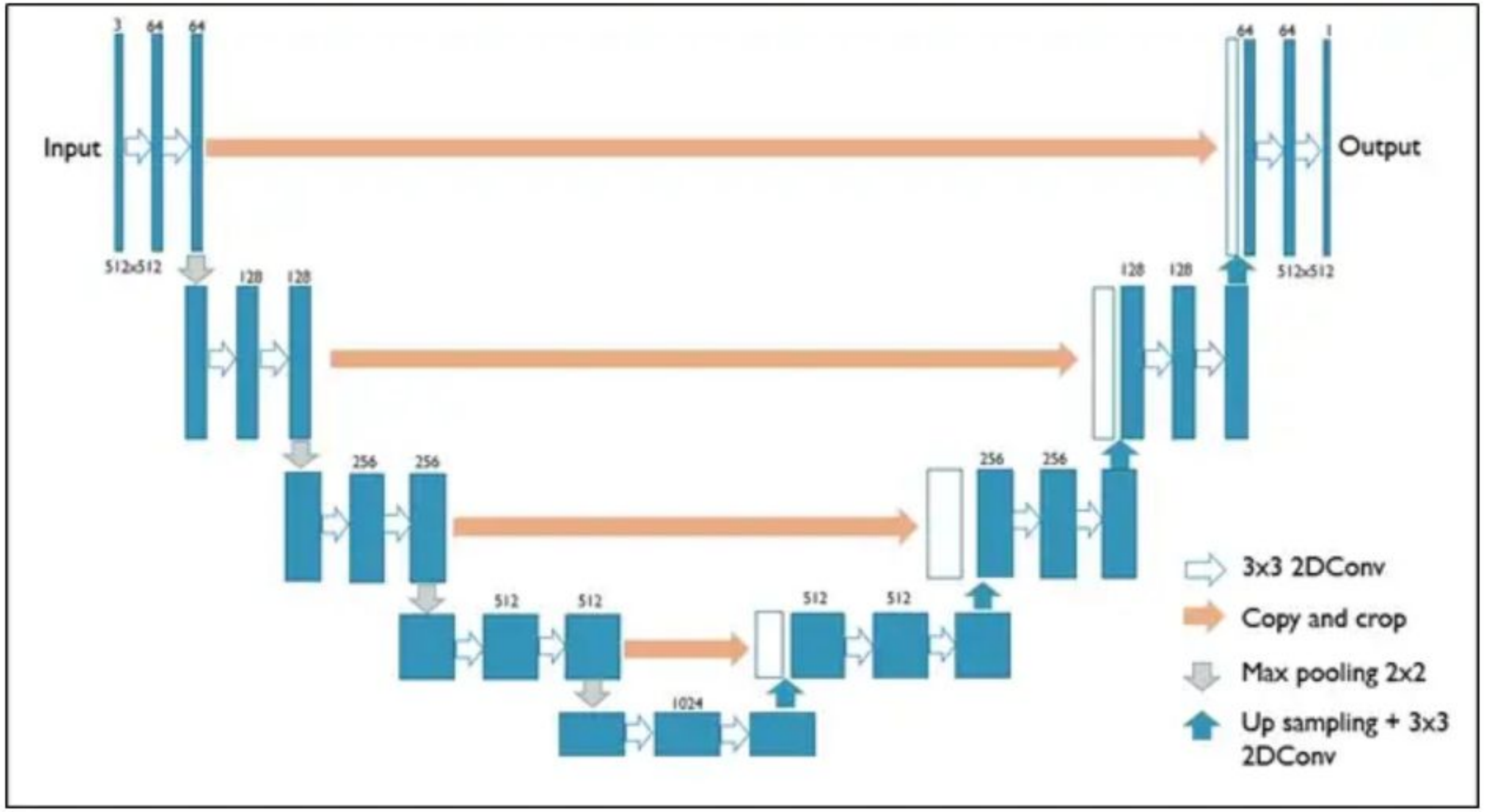
- The input to the model is seismic images after preprocessing
- The required output is a full resolution image depicting the faulty regions on the input
- Segmenting out faulty regions from the input image is an option
- Hence, the fault mapping can be viewed as an image segmentation task





U-Net framework for image segmentation

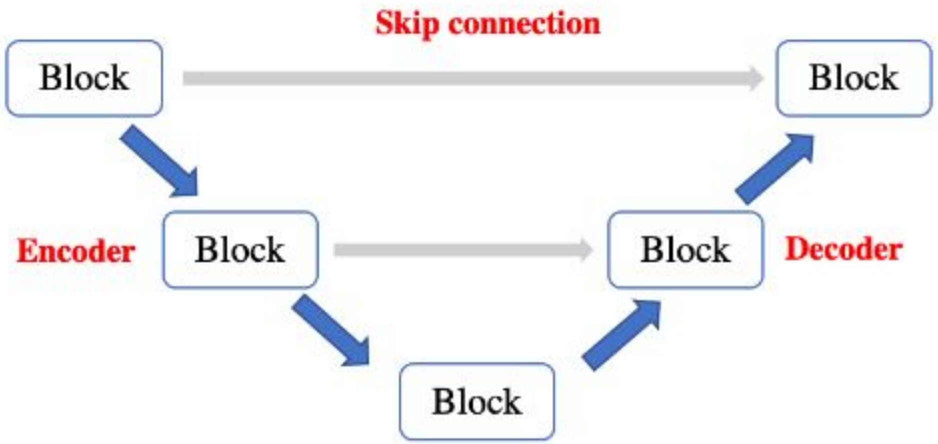
- -Originally developed for segmentation in medical images



U-Net Framework (Image by Rachel Zhiquing Zheng, see Reference)

Features of U-Net

- a forward path of contraction involving several downsampling steps. This is the encoder section of the U-Net which involves two 3x3 convolutions followed by a ReLU, a 2x2 max pooling and a stride of 2 for downsampling.
- Skip Connections
- a reverse path of expansion involving several upsampling steps. This is the decoder section consisting of an upsampling of feature map, a 3x3 convolution and a concatenation of a feature map from the previous contracting block followed by 3x3 convolutions each with ReLU activation.



Initial Basic Implementation of U-Net Model

U-Net Implementation.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

Code + Text

[3]
```python
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```python
class ContractingBlock(nn.Module):
    '''
    Performs two convolutions followed by a max pool operation.
    '''
    def __init__(self, input_channels):
        super(ContractingBlock, self).__init__()

        self.conv1 = nn.Conv2d(input_channels, 2*input_channels, kernel_size=3, padding=(1,1))
        self.conv2 = nn.Conv2d(2*input_channels, 2*input_channels, kernel_size=3, padding=(1,1))
        self.activation = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)

    def forward(self, x):
        '''
        Function for completing a forward pass of ContractingBlock:
        Given an image tensor, completes a contracting block and returns the transformed tensor.
        '''
```

Model Framing as per user annotation

The main objective is to frame U-net model as per user annotation to make user to access the model hyperparameters in real time and to reach optimal accuracy.

The Stepwise approach was guided to achieve the following framework:

1. The Filter Size was considered as variable and other parameters were fixed

2. Number of convolution layers in an encoder block was considered as variable keeping other parameters fixed.

Code Snippets

Initial Attempt to design the custom the custom U-net architecture by implementing ideas:-

1. Made Helper functions (conv_block,encoder_blockand decoder_block)which will help us to add flexibility in model.

Decoder block :-

```
def decoder_block(input,skip_features,num_filters):
  x = UpSampling3D(size = 1)(input)
  x = Conv3DTranspose(num_filters,2,strides=2)(x)
  skip_features = Cropping3D(((4,4), (4,4), (4,4)))(skip_features)
  x = tf.concat([skip_features,x], axis = 4)
  x = conv_block(x,num_filters)

  '''x = UpSampling3D(size = 1)(b1)
  print(x)
  x = Conv3DTranspose(num_filters,2,strides=2)(x)

  skip_features = Cropping3D(((4,4), (4,4), (4,4)))(s[3])
  x = tf.concat([skip_features,x], axis = 4)
  x = conv_block(x,num_filters)
  num_filters = num_filters/2
  print(num_filters)'''

  return x
```

Convolution block :-

```
def conv_block(input,num_filters):
  x = Conv3D(num_filters,3,activation = 'relu')(input)
  #x = BatchNormalization()(x)
  #x = Activation("relu")(x)

  x = Conv3D(num_filters,3,activation = 'relu')(x)
  #x = BatchNormalization()(x)
  #x = Activation("relu")(x)
  return x
```

Encoder block :-

```
def encoder_block(input,num_filters):
  x = conv_block(input,num_filters)
  p = MaxPooling3D(pool_size=(2,2,2))(x)

  return x, p
```

Unet Model without Padding → Incorrect architecture

```
def unet(input_shape,filter_no,no_of_downsamples):
  inputs = Input(input_shape)
  # encoder part
  s=[]#skip
  p=[]#passed
  a,b = encoder_block(inputs,filter_no)
  s.append(a)
  p.append(b)
  for i in range(no_of_downsamples-1):
    filter_no *= 2
    a,b = encoder_block(p[i],filter_no)
    s.append(a)
    p.append(b)

  # Bridge Part
  b1 = conv_block(p[no_of_downsamples-1],filter_no * 2)

  # Decoder part
  #d = []
  #Decoder1
  num_filters = filter_no
  x = UpSampling3D(size = 1)(b1)
  x = Conv3DTranspose(num_filters,2,strides=2)(x)
  print(x)
  skip_features = Cropping3D(((4,4), (4,4), (4,4)))(s[3])
  x = tf.concat([skip_features,x], axis = 4)
  x = conv_block(x,num_filters)
  num_filters = num_filters/2
  print(x)
```

```
  #Decoder2
  x = UpSampling3D(size = 1)(x)
  print(x)
  x = Conv3DTranspose(num_filters,2,strides=2)(x)
  print(x)
  skip_features = Cropping3D(((16,16), (16,16), (16,16)))(s[2])
  print(x)
  x = tf.concat([skip_features,x], axis = 4)
  x = conv_block(x,num_filters)
  num_filters = num_filters/2
  #Decoder1
  x = UpSampling3D(size = 1)(x)
  x = Conv3DTranspose(num_filters,2,strides=2)(x)
  print(x)
  skip_features = Cropping3D(((40,40), (40,40), (40,40)))(s[1])
  x = tf.concat([skip_features,x], axis = 4)
  x = conv_block(x,num_filters)
  num_filters = num_filters/2
  #Decoder1
  x = UpSampling3D(size = 1)(x)
  x = Conv3DTranspose(num_filters,2,strides=2)(x)
  skip_features = Cropping3D(((88,88), (88,88), (88,88)))(s[0])
  x = tf.concat([skip_features,x], axis = 4)
  x = conv_block(x,num_filters)
  num_filters = num_filters/2
  outputs = Conv3D(1,1,activation="sigmoid")(x) #(d[no_of_downsamples-1])
  model = tf.keras.Model(inputs = inputs, outputs = outputs)
  return model
```

2nd attempt - Varying Filter Size

```
#Problems
# 1) Doesn't take in account the loss of dimension while downsampling in case of odd size: No logic on when to stop downsampling. --solved
#    Eg:for downsampling=3, 572-286-"143-142"--
# 2) Error in filter size while concatinating after Upsampling. Eg: [?,286,286,286,128], [?,286,286,286,64] --solved
def unet(input_shape,filter_no,no_of_downsamples):
  inputs = Input(input_shape)
  # encoder part
  s=[]#skipped
  p=[]#passed
  a,b = encoder_block(inputs,filter_no)
  s.append(a)
  p.append(b)
  for i in range(no_of_downsamples-1):
    filter_no *= 2
    a,b = encoder_block(p[i],filter_no)
    s.append(a)
    p.append(b)
  filter_no*=2
    # Bridge Part
    b1 = conv_block(p[no_of_downsamples-1],filter_no)

    # Decoder part
    b2 = decoder_block(b1,s[no_of_downsamples-1],filter_no)
    d = []
    d.append(b2)
    for i in range(no_of_downsamples-1):
      c = decoder_block(d[i],s[no_of_downsamples-i-2],filter_no)
      filter_no/=2
      d.append(c)
    outputs = Conv3D(1,1,padding="same",activation="sigmoid")(d[no_of_downsamples-1])
    model = tf.keras.Model(inputs = inputs, outputs = outputs)
    return model
```

3rd attempt :- Varying # Conv layers

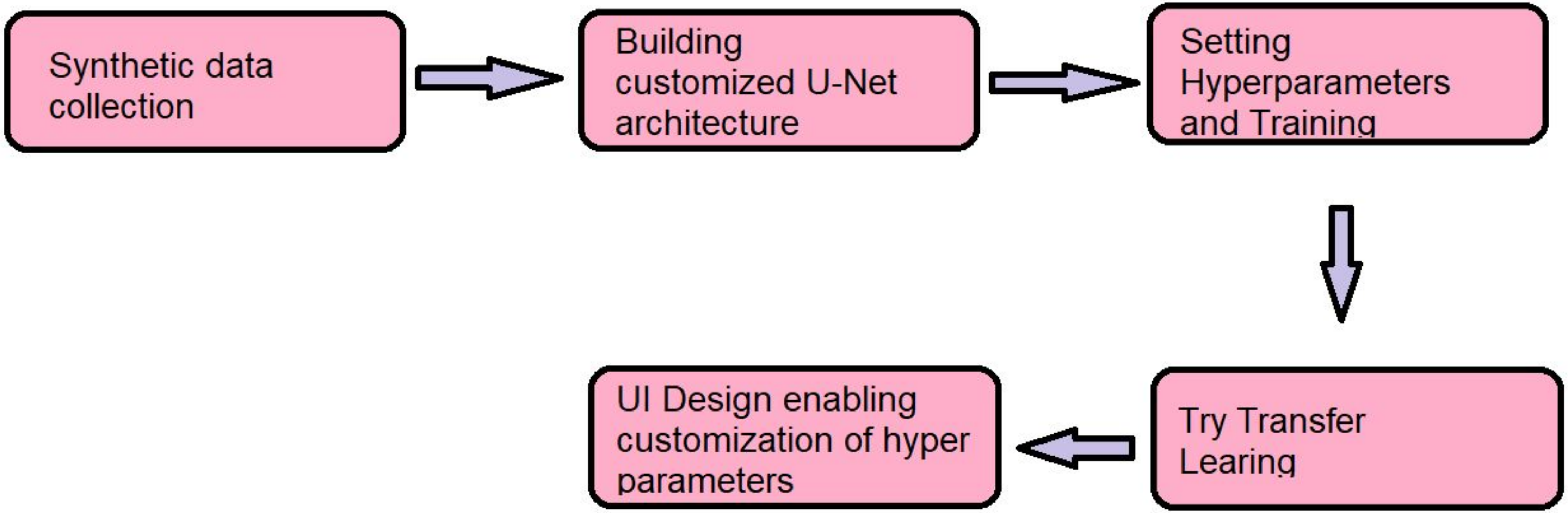
```
#Problems
# 1) Dim error: ValueError: Input 0 of layer sequential_3 is incompatible with the layer: :
# expected min_ndim=4, found ndim=3. Full shape received: [32, 28, 28] --solved
def unet(input_shape,filter_no,no_of_downsamples,num_conv):
  inputs = Input(input_shape)
  # encoder part
  s=[]#skipped
  p=[]#passed
  a,b = encoder_block(inputs,filter_no,num_conv)
  s.append(a)
  p.append(b)
  for i in range(no_of_downsamples-1):
    filter_no *= 2
    a,b = encoder_block(p[i],filter_no,num_conv)
    s.append(a)
    p.append(b)
  filter_no*=2
```

Languages/Libraries used-

-Python (via Jupyter Notebook)
-Tensorflow (for deep learning model)
-Segyio (for handling standard seismic data
-Matplotlib (to plot results)
-Numpy

Numerous Libraries are used in the UI design code files such as tqdm, segpy, ray, sklearn, gurobipy and many more...

Tasks Involved:-



Code snippets:-

The UNET Model consist of encoded part and decoded part

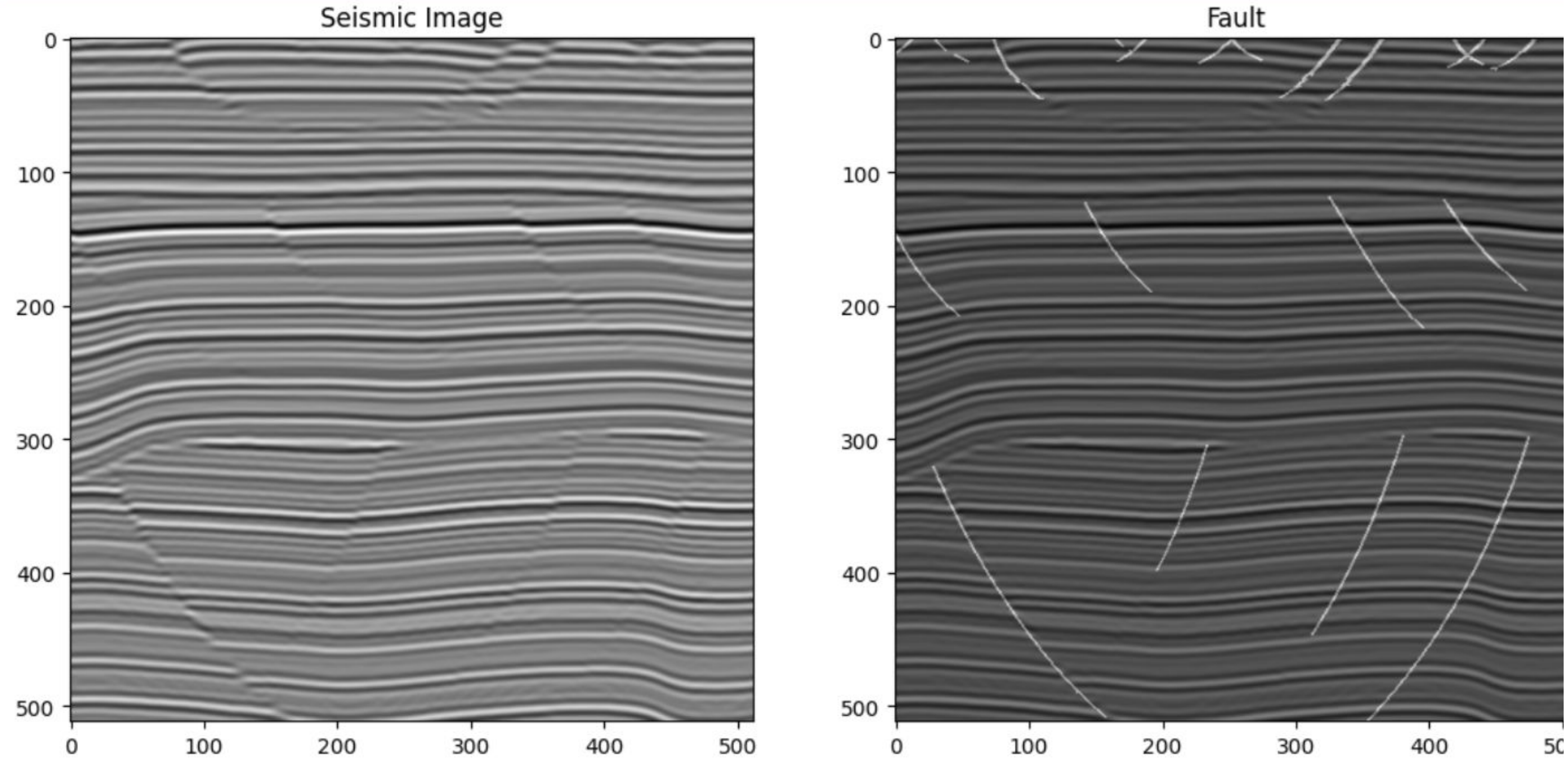
Input images go through 4 contracting layers and 4 expanding layers

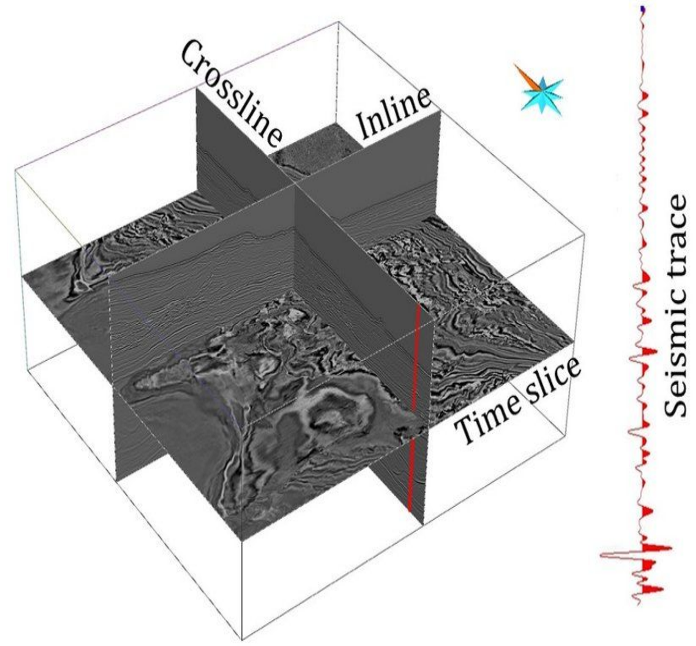
The final output is ensured to be of same size as input (batch_size x 512 × 512)

```python
#combined unet model
class UNet(nn.Module):

    def __init__(self, input_channels, output_channels, hidden_channels=64):
        super(UNet, self).__init__()
        # "Every step in the expanding path consists of an upsampling of the feature map"
        self.upfeature = FeatureMapBlock(input_channels, hidden_channels)
        self.contract1 = ContractingBlock(hidden_channels)
        self.contract2 = ContractingBlock(hidden_channels * 2)
        self.contract3 = ContractingBlock(hidden_channels * 4)
        self.contract4 = ContractingBlock(hidden_channels * 8)
        self.expand1 = ExpandingBlock(hidden_channels * 16)
        self.expand2 = ExpandingBlock(hidden_channels * 8)
        self.expand3 = ExpandingBlock(hidden_channels * 4)
        self.expand4 = ExpandingBlock(hidden_channels * 2)
        self.downfeature = FeatureMapBlock(hidden_channels, output_channels)

    def forward(self, x):

        # Keep in mind that the expand function takes two inputs,
        # both with the same number of channels.
        x0 = self.upfeature(x)
        x1 = self.contract1(x0)
        x2 = self.contract2(x1)
        x3 = self.contract3(x2)
        x4 = self.contract4(x3)

        x5 = self.expand1(x4, x3)
        x6 = self.expand2(x5, x2)
        x7 = self.expand3(x6, x1)
        x8 = self.expand4(x7, x0)
        xn = self.downfeature(x8)
        return xn
```

Plotting inline image from 3-d seismic data data

```
plt.imshow(volumes[0].T, cmap="gray");
```





Training model

Data is feeded into model with the help of torch vision dataloader which stacks image into piles of batch sizes (Here, 1)

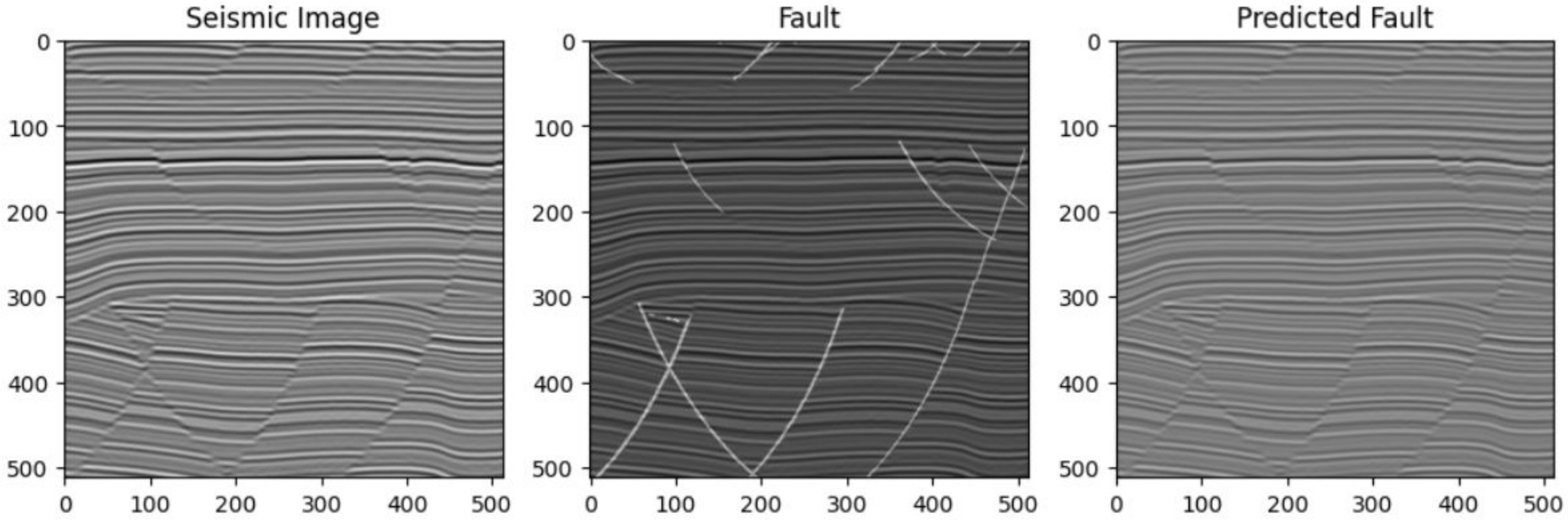
Images are loaded to device memory, loss is calculated with respect to label, backpropagation is done and weights are finally updated for each pile in an epoch.

```
def train():
    dataloader = DataLoader(
        dataset,
        batch_size=batch_size,
        shuffle=True)
    print(dataloader)
    unet = UNet(input_dim, label_dim).to(device)
    unet_opt = torch.optim.Adam(unet.parameters(), lr=lr)
    cur_step = 0
    s=0
    train_losses = []

    for epoch in range(n_epochs):
        s=s+1
        if(s==5):
            break

        for real, labels in tqdm(dataloader):
            cur_batch_size = len(real)
            # Flatten the image
            real = real.to(device)
            labels = labels.to(device)


            ### Update U-Net ###
            unet_opt.zero_grad()
            pred = unet(real)
            #print(pred.shape)
            unet_loss = criterion(pred, labels)
            print(unet_loss)
            l.append(unet_loss)

            unet_loss.backward()
            unet_opt.step()


            if cur_step % display_step == 0:
                print(f"Epoch {epoch}: Step {cur_step}: U-Net loss: {unet_loss.item()}")
                #show_tensor_images(
                    #crop(real, torch.Size([len(real), 1, target_shape, target_shape])),
                    #size=(input_dim, target_shape, target_shape)
                #)
                #show_tensor_images(real.T, size=(input_dim, target_shape, target_shape))
                #show_tensor_images(labels.T, size=(label_dim, target_shape, target_shape))
                #show_tensor_images(torch.sigmoid(pred).T, size=(label_dim, target_shape, target_shape))
                show_tensor_images(real.T, labels.T, torch.sigmoid(pred).T, size=(input_dim, target_shape, target_shape))
            cur_step += 1
```

Prediction at Initial epoch

```
model, pred, loss = train()
```

```
<torch.utils.data.dataloader.DataLoader object at 0x000001D1FCD8B250>
  0%|          | 0/101 [00:00<?, ?it/s]
tensor(0.6502, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>)
Epoch 0: Step 0: U-Net loss: 0.650175154209137
```

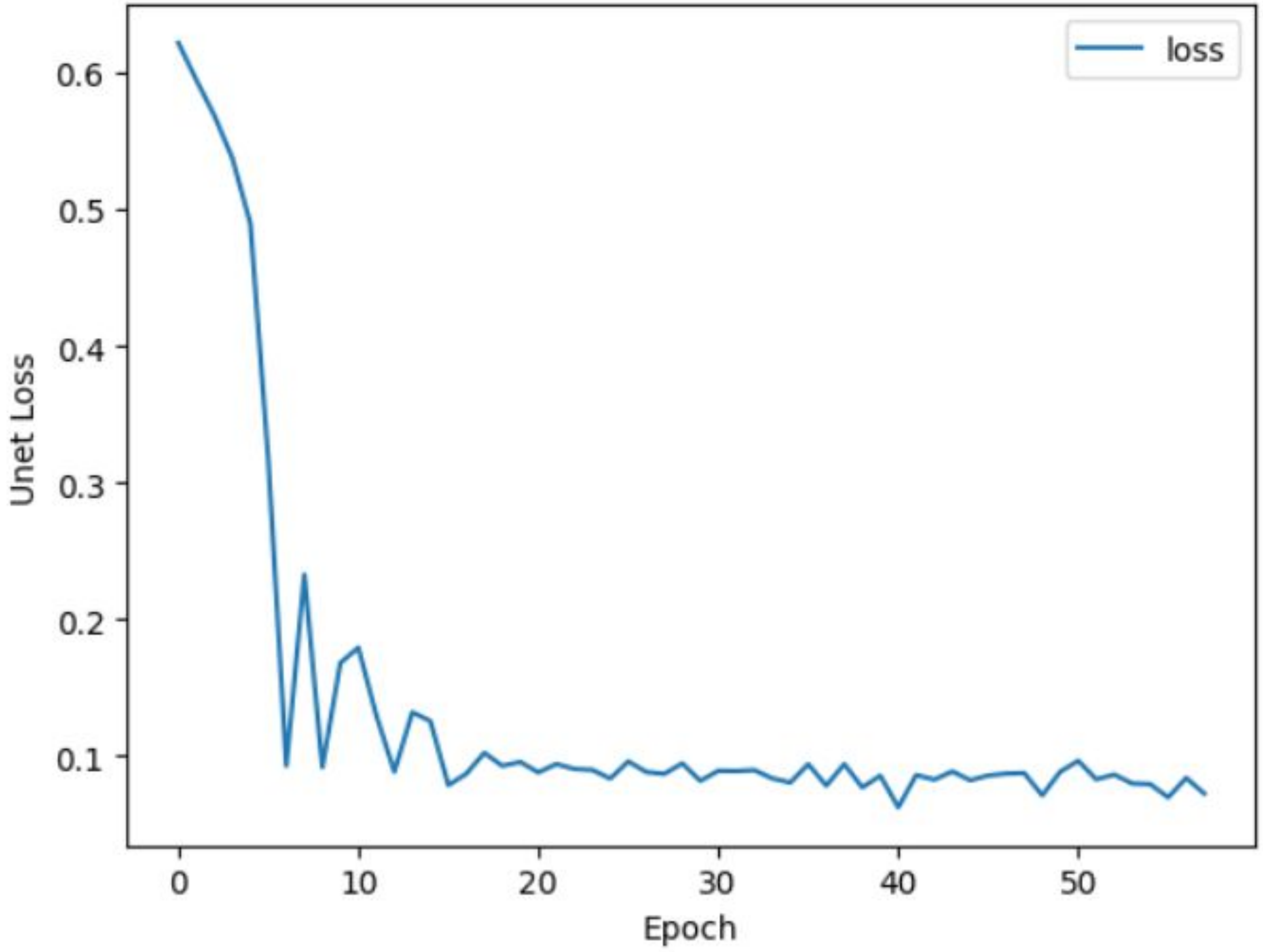



```
tensor(0.6218, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>)
```

Loss curve after training

It is clearly visible that loss curve is gradually decreasing which indicate the model has successfully been trained on the fault detection dataset

The oscillation is due to batch-wise training. Batch wise training never reach global optima. It can only reach near to it.



Timeline

Week 0:-

- Learning ML-DL for image object detection.
- Exploring research papers on fault detection model on seismic images

Week-1:-

- Initial visualization and Exploratory data analysis of the training dataset for fine tuning.
- The Basis of Unet model were understood to design the user-friendly framework.
- The Initial trials were made to make custom U-net network .

Week-2:-

- The stepwise approach was followed to make custom model by considering variable filter size ,convolution layers and keeping rest of the hyperparameters as constant.

Week-3:-

- Started Working on GUI and got familiar with QT Designer and CUDA
- Designed front end panel taking hyperparameters using QT designer

Thank you

~ Team 28