National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Anusha Gorur Chandrashekar
Student ID: x18195059

School of Computing
National College of Ireland

Supervisor: Dr.Paul Stynes, Dr.Pramod Pathak

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Anusha Gorur Chandrashekar
Student ID:	x18195059
Programme:	Data Analytics
Year:	2019-2020
Module:	MSc Research Project
Supervisor:	Dr.Pramod Pathak, Dr.Paul Stynes
Submission Due Date:	17/08/2020
Project Title:	Configuration Manual
Word Count:	1913
Page Count:	14

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:	Anusha Gorur Chandrashekar
Date:	17th August 2020

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 the 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

Anusha Gorur Chandrashekar
x18195059
Msc Data Analytics

Introduction

This configuration manual is a formal document includes sections specifying the hardware, software requirements, design detail, operational information, implementation phases and the settings of the research project in detail: "**A Deep Neural Network Framework for Seismic Image Classification & Analysis**"

1. System Configurations

1.1 Hardware

- **Processor:** Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz, 1800 Mhz, 4 Core(s), 8 Logical Processor(s)
- **RAM:** 16 GB
- **System Type:** Windows OS, 64-bit
- **GPU:** Intel(R) UHD Graphics Family, 8GB
- **Storage:** 1 TB HDD

1.2 Software

- **Google Colaboratory:** A promising machine learning research platform that offers free cloud service to run machine learning and deep learning models. The interface is similar to Jupyter notebook with the default installed libraries also with hardware accelerator options such as GPU, TPU, and runtime type. The "out of memory" is an infamous warning while working with a large volume of the database, hence we need expensive GPU memory. The Google Colab provides free Tesla K80 GPU of about 12GB and Tensor Processing Unit (TPU) which is 15-30 times faster than GPU.

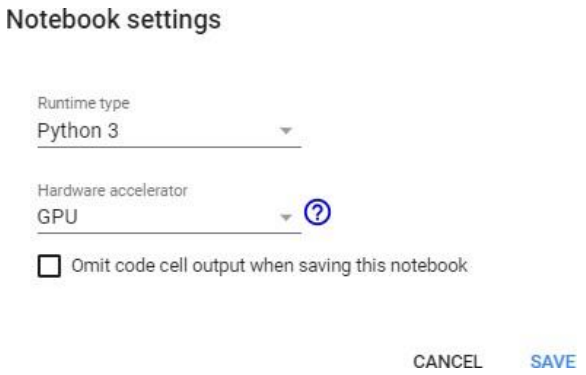


Figure 1: GPU Configuration: Google Colaboratary

- **Microsoft Excel:** A spreadsheet program offered by Microsoft is used for visualization of data, plots, table formation.

2. Project Development

Implementation was carried out entirely using python programming. This research work mainly has three phases namely: Data preparation, Modelling stage, Evaluation stage. The first stage has data preprocessing, data selection followed by modelling stage includes model implementation using TensorFlow, Keras, Scikit-

learn, fine-tuning hyperparameters and lastly, evaluation models using performance metrics such as MAE, precision, categorical accuracy.

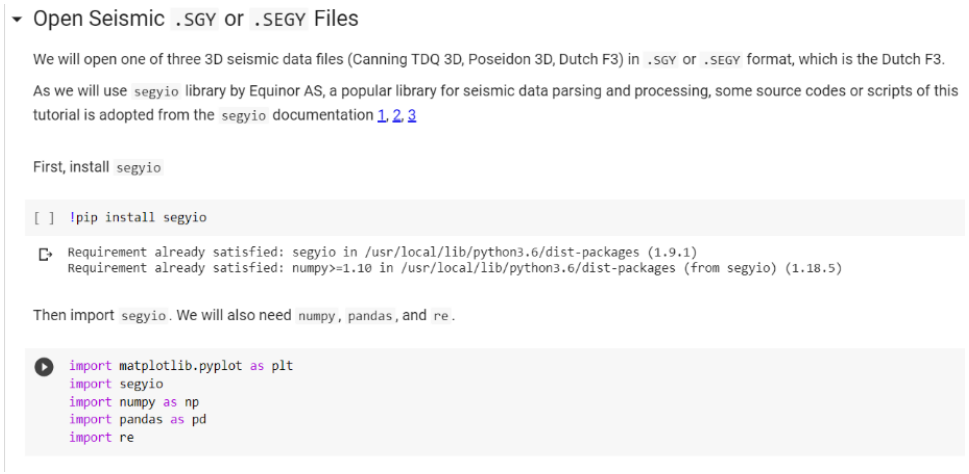
2.1 Data Preparation

Importing of datasets and data manipulation are done using pandas (data frames), NumPy (arrays), segyio (to import seismic volumes. The below sections describe the data pre-processing stages employed in this research work.

2.1.1 F3 Netherlands Seismic Dataset

The F3 Netherlands seismic data is available publicly in the DGB Earth Sciences repository. Using segy.io to import the seismic data SEGY files format from the F3 Netherlands directory. Segyio library is popular for seismic data parsing and processing. And some of the code sources are adopted from segyio documentation.

- Manipulating the similarity to enhance discontinuity/faults of the seismic volume.
- creating fault mask, fault extraction, displaying samples of amplitude time slices.
- normalizing the plot by calculating the percentile of its amplitude.
- Creating data slices from the seismic cube.




```
import numpy as np
import pandas as pd
!pip install obspy
!pip install keras_tqdm
import obspy

import keras
import time
from keras_tqdm import TQDMNotebookCallback
from tqdm import tnrange, tqdm_notebook
import matplotlib.pyplot as plt


import tensorflow as tf
from obspy.io.segy.segy import _read_segy
from sklearn.model_selection import train_test_split

np.random.seed(42)
%matplotlib notebook
```

Figure 1- Importing libraries

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercor

Enter your authorization code:
··········
Mounted at /content/drive
```

```
[ ] cd /content/drive/'My Drive'/'Public geoscience Data'

/content/drive/.shortcut-targets-by-id/0B7brcf-eGK8CRUhfRW9rSG91bW8/Public geoscience Data
```

Code `ls -l` to see what's inside the `Public geoscience Data` directory.

Below is a summary of the contents.

Folder	Content	File extensions
48 well composite logs in machine readable format	one ZIP file	.zip
Canning 3D TDQ	3D Seismic	.sgy
collection of geological images for classification	Geoscience figures	.png, .jpg
Dutch F3 seismic data	3D seismic	.segy
F3 seismic data plus classification data for machine learning	3D seismic, Python script	.segy, .py
GEOLINK North sea wells with Lithology interpretation	Lithology and wells, Excel spreadsheet	.zip, .xlsx, .png
geolocate documents with shapefiles	one ZIP file	.zip
Poseidon Seismic and well data NW Shelf Australia	Core images, Excel spreadsheet, 3D seismic, report	.segy, .las, .xlsx, .pdf
public Core images and core interpretation	Core descriptions	.zip, .las, .pdf, .tiff, .jpg
RealRocks RealPore RealLogs Public Geoprovider sponsored	ZIP files, Excel spreadsheet	.zip, .xlsx
reports for images	Documents, reports	.pdf

Figure 2- Importing data from drive and directory content display

Our purpose now is to visualize the Dutch F3. It's important that we know the range of the **inline** and the **crossline**. We also need to normalize our plot later by calculating the **percentile of its amplitude**.

```
[ ] with segyio.open(filename) as f:
      print('Inline range from', min(f.ilines), 'to', max(f.ilines))
      print('Crossline range from', min(f.xlines), 'to', max(f.xlines))
      data = f.trace.raw[:]
      clip_percentile = 99
      vm = np.percentile(data, clip_percentile)

    f'The {clip_percentile}th percentile is {vm:.0f}; the max amplitude is {data.max():.0f}'

Inline range from 100 to 750
Crossline range from 300 to 1250
'The 99th percentile is 6517; the max amplitude is 32767'
```

Figure 3- Normalizing plot by calculating percentile of amplitude

2.1.2 MalenoV labeled Dataset

MalenoV contains facies annotation of the 3D SEGY seismic cube which was previously defined by the user. The data contains 158812 rows and 4 columns Inline, Crossline, Time, Labels/Class. The first column, 339 inline corresponds to Crossline 330 to 1247, time slices ranging from 716 to 940, and appropriate facies label is predicted.

```
[ ] labels = pd.read_csv('/content/drive/My Drive/classificationlabel.ixz', delimiter=" ", names=["Inline","Xline","Time","Class"])
    labels.describe()
    print(labels)
```

```
        Inline  Xline  Time  Class
0          339    330   716      0
1          339    330   712      0
2          339    330   668      0
3          339    330   672      0
4          339    330   676      0
...        ...    ...   ...    ...
158807     339   1247   924      8
158808     339   1247   928      8
158809     339   1247   932      8
158810     339   1247   936      8
158811     339   1247   940      8

[158812 rows x 4 columns]
```

Figure 4- Loading MalenoV dataset

2.2 Data Selection/Patch Extraction

The F3 seismic dataset is a cube formulated image data, consisting of attributes inline, crossline, time. This data is fed into the model using a python library called segyio. The experiment is conducted on specific slices of seismic cube based on the rich information found in the seismic image slices. Analyzing the seismic image slices has been done using the customized widget Graphical User Interface (GUI) created using python. An interactive 2D and 3D viewer interface are developed to analyze the image.

```
def seis_interact(cube=data, type='il', il_loc=400, xl_loc=1000, twt_loc=1404,
                  il_array=inlines, xl_array=crosslines, twt_array=twt,
                  cmap='gray', vmin=-vm, vmax=vm):
    """
    Interactive seismic viewer
    """

    if type == 'il':
      a_line = il_array
      b_line = xl_array
      c_line = twt_array
      loc = il_loc

    if type == 'xl':
      a_line = xl_array
      b_line = il_array
      c_line = twt_array
      loc = xl_loc

    if type == 'ts':
      a_line = twt_array
      b_line = il_array
      c_line = xl_array
      loc = twt_loc

    slices = slicing(cube, type, loc, a_line)
    display_slice(slices, type, b_line, c_line, cmap, vmin, vmax)
```

Figure 5- Interactive 2D viewer GUI

```
seis_interact(cube=data, type='xl', il_loc=401, xl_loc=800, twt_loc=1208,
              il_array=inlines, xl_array=crosslines, twt_array=twt,
              cmap='gray', vmin=-vm, vmax=vm)
```

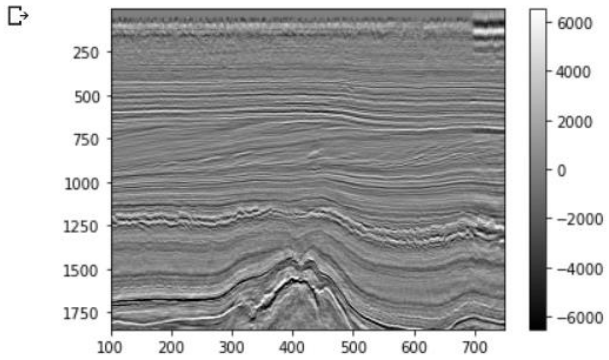


Figure 6- Interactive seismic image slice

A graphical object is created using the scale widget, which enables the user to optimize a numerical value by adjusting the knob along the scale of values assigned. A slider is created using the scale() method. Parameters have set to minimum and maximum values to inline, crossline, timeslice. Using matplotlib colormaps are built and parameters are set to gray, seismic, RdBu, PuOr to enhance the faults, explicitly view the amplitudes in the seismic image.

```
plt.xlabel('Inline'); plt.ylabel('Crossline')
plt.gca().xaxis.set_ticks_position('top') # axis on top
plt.gca().xaxis.set_label_position('top') # label on top
plt.xlim(min(inlines), max(inlines))
plt.ylim(min(crosslines), max(crosslines))
plt.axis('equal')
plt.show()
```

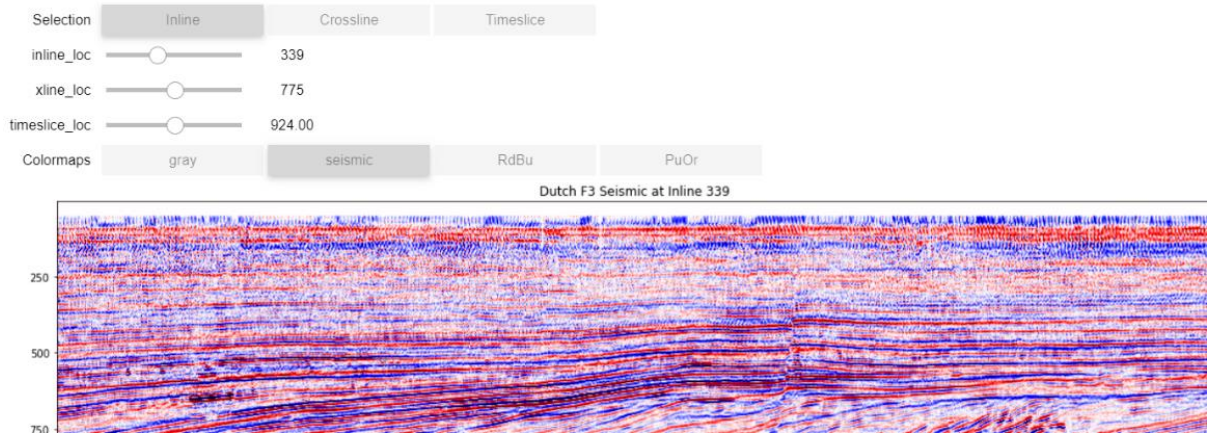


Figure 7- Interactive 3D viewer

It is now easy to select the slices which are more insightful based on the analysis done using GUI viewer. The next step is to slice the desired images from the seismic cube to perform further analysis. In the below code snippet it is evident that slice 500th and 339th are selected for the interpretation of seismic facies. The data is read in 212.3 seconds and slices are created in 6.2 seconds.

```
[ ] filename = '/content/drive/My Drive/Public geoscience Data/Dutch F3 seismic data/Dutch Government_F3_entire_8bit seismic.segy'

    t0=time.time()
    stream0 = _read_segy(filename, headonly=True)
    print('--> data read in {:.1f} sec'.format(time.time()-t0))

    t0=time.time()

    labeled_data = np.stack(t.data for t in stream0.traces if t.header.for_3d_poststack_data_this_field_is_for_in_line_number == 339).T
    inline_data = np.stack(t.data for t in stream0.traces if t.header.for_3d_poststack_data_this_field_is_for_in_line_number == 500).T
    xline_data = np.stack(t.data for t in stream0.traces if t.header.for_3d_poststack_data_this_field_is_for_cross_line_number == 500).T

    print('--> created slices in {:.1f} sec'.format(time.time()-t0))

--> data read in 212.3 sec
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2822: FutureWarning: arrays to stack must be passed as a "sequence" type
  if self.run_code(code, result):
--> created slices in 6.2 sec
```

Figure 8- data slicing

The patch extraction is carried out to extract the patches from the 2-dimensional, 3-dimensional array. A sliding window approach is implemented to extract patches from each pixel in an image.

```
def patch_extractor2D(img,mid_x,mid_y,patch_size,dimensions=1):
    try:
        x,y,c = img.shape
    except ValueError:
        x,y = img.shape
        c=1
    patch= np.pad(img, patch_size//2, 'constant', constant_values=0)[mid_y:mid_y+patch_size,mid_x:mid_x+patch_size]
    if c != dimensions:
        tmp_patch = np.zeros((patch_size,patch_size,dimensions))
        for uia in range(dimensions):
            tmp_patch[:,:,uia] = patch
        return tmp_patch
    return patch
image=np.random.rand(10,10)//.1
print(image)

patch_extractor2D(image,10,10,4,1)

[[ 3.  9.  7.  5.  1.  1.  0.  8.  6.  7.]
 [ 0.  9.  8.  2.  1.  1.  3.  5.  4.  2.]
 [ 6.  1.  2.  3.  4.  7.  1.  5.  5.  0.]
 [ 6.  1.  0.  9.  9.  8.  3.  0.  6.  4.]
 [ 1.  4.  0.  9.  2.  6.  3.  5.  5.  1.]
 [ 9.  7.  9.  8.  5.  9.  0.  1.  0.  3.]
 [ 3.  2.  8.  3.  2.  5.  1.  8.  0.  9.]
 [ 7.  1.  0.  8.  7.  7.  7.  0.  3.  1.]
 [ 8.  6.  3.  0.  3.  3.  7.  6.  8.  4.]
 [ 1.  7.  7.  5.  7.  4.  5.  4.  0.  1.]]
array([[ 8.,  4.,  0.,  0.],
```

Figure 9- Patch extraction of an image

3. Modelling

3.4.1 Data split

Train Test split is performed and random_state is assigned to 42, to get the same split of train and test data points every time. Used for reproducing the same problem every time it is run. If a random_state is not used in train_test_split, the split obtained will be a different set of train and test data points and will not help in debugging in case you get an issue.

```
In [15]: train_data, test_data, train_samples, test_samples = train_test_split(
             labels, sampler, random_state=42)
         print(train_data.shape,test_data.shape)

         (119109, 4) (39703, 4)
```

Figure 10- Data Split

This is the Keras data generator that wraps the patch_extractor2D()

```
[ ]  class SeismicSequence(keras.utils.Sequence):
        def __init__(self, img, x_set, t_set, y_set, patch_size, batch_size, dimensions):
            self.slice = img
            self.X,self.t = x_set,t_set
            self.batch_size = batch_size
            self.patch_size = patch_size
            self.dimensions = dimensions
            self.label = y_set

        def __len__(self):
            return len(self.X) // self.batch_size

        def __getitem__(self,idx):
            sampler = np.random.permutation(len(self.X))
            samples = sampler[idx*self.batch_size:(idx+1)*self.batch_size]
            labels = keras.utils.to_categorical(self.label[samples], num_classes=9)
            if self.dimensions == 1:
                return np.expand_dims(np.array([patch_extractor2D(self.slice,self.X[x],self.t[x],self.patch_size,self.dimensions) for x in samples]),
            else:
                return np.array([patch_extractor2D(self.slice,self.X[x],self.t[x],self.patch_size,self.dimensions) for x in samples]), labels
```

Figure 11- Keras data generator

Then we build acc_assess() to format our test accuracy assessment for each model

```
def acc_assess(data,loss=['categorical_crossentropy'],metrics=['acc']):
    if not isinstance(loss, list):
        try:
            loss = [loss]
        except:
            raise("Loss must be list.")
    if not isinstance(metrics, list):
        try:
            metrics = [metrics]
        except:
            raise("Metrics must be list.")
    out='The test loss is {:.3f}\n'.format(data[0])
    for i, metric in enumerate(metrics):
        if metric in 'mae':
            out += "The total mean error on the test is {:.3f}\n".format(data[i+1])
        if metric in 'accuracy':
            out += "The test accuracy is {:.1f}%\n".format(data[i+1]*100)
    return out
print(acc_assess([1,2,3],'bla',["acc", "mae"]))
```

Figure 12- acc_assess function

3.4.2 Hyperparameters-tuning

In neural networks, the hyperparameters tuning is done using TensorFlow to the training job. To gain the advantage of Keras Early stopping, the training code should report to the hyperparameter metric AI platform training persistently. When the monitored metrics have stopped showing improvements, the training should be stopped hence we use early stopping. The arguments considered are “min_delta”- to ignore the improvement less than the assigned value i.e. 0. “verbosity” mode, mode specifying to “auto” to stop the training, when monitored metrics are not increasing. “Checkpoint” to store the best weights only. Below is the Keras API reference to Early stopping. The weights to be stored in python “HDF5” file in the binary data format.

```
earlystop1 = keras.callbacks.EarlyStopping(monitor='val_loss',
                                min_delta=0,
                                patience=10,
                                verbose=0, mode='auto')

earlystop2 = keras.callbacks.EarlyStopping(monitor='val_acc',
                                min_delta=0,
                                patience=15,
                                verbose=0, mode='auto')

checkpoint_path = "weights/weights3.best.hdf5"

checkpoint = keras.callbacks.ModelCheckpoint(checkpoint_path,
                                     monitor='val_loss',
                                     verbose=0,
                                     save_best_only=True,
                                     save_weights_only=True,
                                     mode='auto',
                                     period=1)

callbacklist = [TQDMNotebookCallback(leave_inner=True, leave_outer=True), earlystop1, earlystop2, checkpoint]
```

Figure 13- Keras EarlyStopping class

Some parameters need to be defined to fit the model we are testing. The "patch_size" of the image is set to 64 in the case of VGG, 244 for ResNet model, and EfficientNet model, "batch_size" is set to 64, the number of image data samples to be processed before the model is updated also it controls the error gradient estimation. The best starting point is generally 32 or 64, channels refer to the number of patches stacked in the image volume, an "epoch" is set to 10-15 indicating the number of times or the iteration the entire training dataset is passed into the model, argument "steps" are assigned to 200 to 450, meaning that in one step, "batch_size" = 64 many samples are processed.

```
patch_size = 64 # for ResNet50 put 244, vgg -64
batch_size = 64
num_channels = 3
num_classes = 9
all_examples = 158812
num_examples = 7500
epochs = 3
steps=200
sampler = list(range(all_examples))
opt = 'adam'
lossfkt = ['categorical_crossentropy']
metrica = ['mae','acc',m]

from keras.optimizers import SGD
sgd = SGD(lr=1e-4, decay=1e-6, momentum=0.9, nesterov=True)
vgg.compile(loss=lossfkt,
                  optimizer=sgd,
                  metrics=metrica)
```

Figure 14- Tuning parameters

The optimizer SGD is chosen for all three models also Adadelta is chosen as an additional optimizer to the VGGNet-16 model, categorical cross-entropy is used as the loss function to measure the performance of the multi-class classification model. The arguments passed to the SGD class are learning rate= 1E-1 to 1E-4 to indicate the step size while optimizing to minimize a loss function, momentum is set to 0.9 to improve the accuracy and faster training.

```
x = base_model.output
x = Flatten()(x)
x = Dense(256,name = 'dense_layer1')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(.5)(x)
x = Dense(num_classes, name = 'pre-softmax_layer')(x)
x = BatchNormalization()(x)
x = Activation('softmax')(x)

vgg = Model(input=base_model.input, output=x)
```

Figure 15- Baseline model

The fully convolutional network is built as our baseline model. However, transfer learning helps in using pre-trained networks and replace the fully connected layers with the baseline model. These fully-connected layers are for classification and specific to the task and fine-tuned according to the seismic data. The main models i.e VGGNet, ResNet, EfficientNet, and the baseline model are designed using the same network blocks, trained using the same optimizers and hyperparameters techniques. The softmax activation last layer is used in the probability fo the classification.

3.4.3 VGGNet 16

The VGGNet-16 network is implemented in this research, 3 x 3 convolutional layers stacked with each other on top, increasing the depth, same padding, and max pool layer of 2 x 2 filter and stride 2. The network is pretty large with 138 million parameters and the last two-fully connected layers followed by the softmax layer for the classification out. The model is specified as sequential and the below explains the architecture.

```
model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool size=(2,2),strides=(2,2)))
```

Figure 16- VGGNet-16 model

```
t0 = time.time()

vgg_hist = vgg.fit_generator(
    SeismicSequence(
        labeled_data,
        train_data["Xline"].values,
        train_data["Time"].values,
        train_data["Class"].values,
        patch_size,
        batch_size,
        3),
    steps_per_epoch=steps,
    validation_data = SeismicSequence(
        labeled_data,
        test_data["Xline"].values,
        test_data["Time"].values,
        test_data["Class"].values,
        patch_size,
        batch_size,
        3),
    validation_steps = len(test_data)//batch_size,
    epochs = epochs,
    verbose = 0,
    callbacks = callbacklist)

print('--> Training for VGG transfer took:'.format(time.time()-t0))
```

```
Training: 100% 10/10 [06:10<00:00, 37.05s/it]
```

Figure 17- Training VGGNet

3.4.4 ResNet

ResNet variants 101 and 152 versions are implemented based on the concept of skip connection. This model is inspired by VGGNet and the network has 101 and 152 layers. The model architecture is adopted from the Imagenet project and mounted on the baseline model. The input fed is of 244 x 244 size, the same hyperparameters tuning is done as VGGNet with SGD optimizer.

```
# from keras.applications import ResNet50 as Resnet
from keras.applications import ResNet101 as Resnet           # Uncomment this line to train ResNet101
# from keras.applications import ResNet152 as Resnet          # Uncomment this line to train ResNet152
# from keras.applications import ResNet50V2 as Resnet          # Uncomment this line to train ResNet50V2
img_size = 64

input_tensor = Input(shape=(img_size,img_size,3))
res_base = Resnet(include_top=False, weights='imagenet', input_tensor=input_tensor, input_shape=None, pooling=None)

for layer in res_base.layers[:-7]:
    layer.trainable = False



q = res_base.output
q = Flatten()(q)
q = BatchNormalization()(q)
q = Activation('relu')(q)
q = Dense(10,name = 'attribute_layer')(q)
q = BatchNormalization()(q)
q = Activation('relu')(q)
q = Dense(num_classes, name = 'pre-softmax_layer')(q)
q = BatchNormalization()(q)
q = Activation('softmax')(q)

resnet = Model(input=res_base.input, output=q)
resnet.summary()
```

```
conv4_block14_3_conv (Conv2D)   (None, 4, 4, 1024)   263168      conv4_block14_2_relu[0][0]
```

Figure 18- ResNet model

3.4.5 EfficientNet B7

Compared to other models, EfficientNet is smaller which only takes 5,330,564 parameters, where it outperforms models with 23 million parameters. To implement our classification model, to hold on top of the efficientNet model we use, GlobalMaxPooling2D to transform 4D to 2D. To preserve the knowledge of the transfer learning model, we freeze the convolutional base's weights. Below is the model summary

```
top_bn (BatchNormalization)     (None, 2, 2, 2560)   10240      top_conv[0][0]
top_activation (Activation)     (None, 2, 2, 2560)   0          top_bn[0][0]
flatten_1 (Flatten)             (None, 10240)        0          top_activation[0][0]
batch_normalization_1 (BatchNor (None, 10240)        40960      flatten_1[0][0]
activation_1 (Activation)       (None, 10240)        0          batch_normalization_1[0][0]
attribute_layer (Dense)         (None, 10)           102410     activation_1[0][0]
batch_normalization_2 (BatchNor (None, 10)           40         attribute_layer[0][0]
activation_2 (Activation)       (None, 10)           0          batch_normalization_2[0][0]
pre-softmax_layer (Dense)       (None, 9)            99         activation_2[0][0]
batch_normalization_3 (BatchNor (None, 9)            36         pre-softmax_layer[0][0]
activation_3 (Activation)       (None, 9)            0          batch_normalization_3[0][0]
==================================================================================================
Total params: 64,241,225
Trainable params: 63,909,987
Non-trainable params: 331,238
```

Figure 19- EfficientNet summary

4. Evaluation

The model evaluation is done based on the classification metrics “precision” and regression metrics “MAE”. Below are the sample figures witnessed as results and evaluation.

```
# Validation
eff_net_score = ef_net.evaluate(np.array([patch_extractor2D(labeled_data,labels["Xline"][x],
                                                            labels["Time"][x],64,3) for x in test_samples]),
                                keras.utils.to_categorical(labels["Class"][test_samples], num_classes=9))
```

```
39703/39703 [==============================] - 164s 4ms/step
```

```
print(acc_assess(eff_net_score, loss_function, ["acc", "mae"]))
```

```
The test loss is 0.043
The test accuracy is 99.7%
The total mean error on the test is 0.008
```

Figure 20- Validation evaluation

```
%matplotlib inline

patch_size = 64
t_max, y_max = xline_data.shape
half_patch = patch_size//2

eff_net_predx = np.full_like(xline_data,-1)
for space in tqdm_notebook(range(y_max),desc='Space'):
    for depth in tqdm_notebook(range(t_max),leave=False, desc='Time'):
        eff_net_predx[depth,space] = np.argmax(ef_net.predict(np.expand_dims(patch_extractor2D(xline_data,space,depth,patch_size,3), axis=0)))

np.save('eff_net_predi.npy', ef_net_predx, allow_pickle=False)
plt.imshow(eff_net_predx)
```

```
Time: 100% 462/462 [00:26<00:00, 17.58it/s]
Time: 100% 462/462 [00:26<00:00, 17.42it/s]
Time: 100% 462/462 [00:27<00:00, 17.57it/s]
Time: 100% 462/462 [00:27<00:00, 16.22it/s]
Time: 100% 462/462 [00:27<00:00, 16.45it/s]
Time: 100% 462/462 [00:27<00:00, 17.40it/s]
Time: 100% 462/462 [00:27<00:00, 15.65it/s]
Time: 100% 462/462 [00:27<00:00, 17.06it/s]
Time: 100% 462/462 [00:27<00:00, 15.89it/s]
Time: 100% 462/462 [00:27<00:00, 16.00it/s]
```

Figure 21- Plotting from NumPy array to image

```
np.save('eff_net_predx.npy', eff_net_predx, allow_pickle=False)
plt.imshow(eff_net_predx)
```

```
<matplotlib.image.AxesImage at 0x7f2c8b224d30>
```

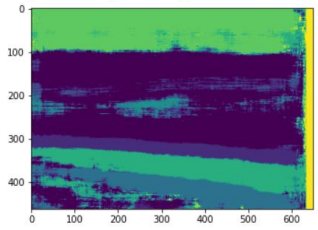


Figure 22- Image plot of EfficientNet B7

```
# summarize history for loss
plt.plot(vgg_hist.history['loss'])
plt.plot(vgg_hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

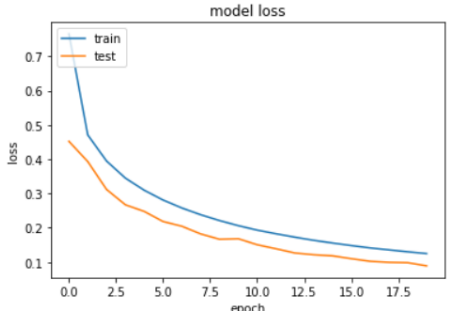


Figure 20- Model loss plot

```
[50] print('--> Training for VGG transfer took:'                    .format(time.time()-t0))
```

```
Training: 100% 10/10 [15:13<00:00, 91.39s/it]
[loss: 0.421, mae: 0.070, acc: 0.961, precision_1: 0.9460 ... val_loss: 0.041, val_acc: 0.998, val_precision_1: 0.996] : 100%
[loss: 0.401, mae: 0.067, acc: 0.967, precision_1: 0.9460 ... val_loss: 0.036, val_acc: 0.999, val_precision_1: 0.996] : 100%
[loss: 0.385, mae: 0.065, acc: 0.967, precision_1: 0.9460 ... val_loss: 0.034, val_acc: 0.998, val_precision_1: 0.996] : 100%
[loss: 0.379, mae: 0.064, acc: 0.967, precision_1: 0.9460 ... val_loss: 0.031, val_acc: 0.998, val_precision_1: 0.997] : 100%
[loss: 0.366, mae: 0.062, acc: 0.964, precision_1: 0.9470 ... val_loss: 0.029, val_acc: 0.999, val_precision_1: 0.997] : 100%
[loss: 0.343, mae: 0.059, acc: 0.970, precision_1: 0.9470 ... val_loss: 0.032, val_acc: 0.999, val_precision_1: 0.997] : 100%
[loss: 0.345, mae: 0.059, acc: 0.967, precision_1: 0.9470 ... val_loss: 0.025, val_acc: 0.999, val_precision_1: 0.997] : 100%
[loss: 0.320, mae: 0.055, acc: 0.970, precision_1: 0.9470 ... val_loss: 0.030, val_acc: 0.998, val_precision_1: 0.997] : 100%
[loss: 0.313, mae: 0.054, acc: 0.972, precision_1: 0.9470 ... val_loss: 0.023, val_acc: 1.000, val_precision_1: 0.997] : 100%
[loss: 0.303, mae: 0.053, acc: 0.974, precision_1: 0.9470 ... val_loss: 0.024, val_acc: 0.999, val_precision_1: 0.997] : 100%

--> Training for VGG transfer took:
```

Figure 21- VGGNet model output

References

[1]"Python2 Tutorial: Sliders in Tkinter", *Python-course.eu*, 2020. [Online]. Available: https://www.python-course.eu/tkinter_sliders.php. [Accessed: 15- Aug- 2020].

[2]"Module: tf.keras | TensorFlow Core v2.3.0", *TensorFlow*, 2020. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras. [Accessed: 15- Aug- 2020].

[3]"Netherlands Dataset: A New Public Dataset for Machine Learning in Seismic Interpretation", *DeepAI*, 2020. [Online]. Available: https://deepai.org/publication/netherlands-dataset-a-new-public-dataset-for-machine-learning-in-seismic-interpretation. [Accessed: 15- Aug- 2020].

[4]J. El Zini, Y. Rizk and M. Awad, "A Deep Transfer Learning Framework for Seismic Data Analysis: A Case Study on Bright Spot Detection", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 5, pp. 3202-3212, 2020. Available: 10.1109/tgrs.2019.2950888.