Imperial College London
Msc in Applied Computational Science and Engineering
Module 9: Independent Research Project
Author: Zongpeng Chen
GitHub Alias: czp21c
Email: zongpeng.chen18@imperial.ac.uk
Date: August 30, 2019

Supervised Seismic Section Classification with Convolutional Neural Networks (CNNs)

Supervisors: Olivier Dubrule, Lukas Mosser
Company: Perenco Oil & Gas
Address: 8 Hanover Square, Mayfair, London W1S 1HQ
Company Supervisors: Jeremy Fortun, Kurt Rattansingh

Abstract:

Artificial acoustic waves penetrating the ground bring some precious information about the subsurface architecture back to the surface. The reflectivity is one of the raw measurements obtained from seismic acquisition and best suited for imaging the subsurface features that helps Oil & Gas Industry choose targets to drill. Features extraction from reflectivity are also common practice in this Industry and this study aims to include them.

All together, these different features usually relate to different geological structures, which usually take months for an expert's team to accurately interpret. There have been a lot of efforts made by geoscientists who tried to optimize this interpretation process by using computer. However, the interpretation accuracy from machine is a big issue and a tradeoff between accuracy and time is hard to determine as well.

This study tries to solve this problem by comparing 3 different Neural Networks (1 Fully Connected and 2 Convolutionals) on 5 labelled seismic sections only to classify 7 different geological units already interpreted by a geophysicist. The location chosen by Perenco to apply this approach is the offshore of Gabon, Africa.

After fine tuning the 3 pipelines separately, the 2 most promising ones, mini-patch CNN model and encoder-decoder model, are combined into a final ensemble model.

As metrics, both the accuracies of the models on test seismic section (93-97%) and predicted image geological rendering are deemed satisfactory.

Keywords:
Seismic Interpretation; Image Classification; Image Segmentation; CNN; GPU; Gabon

1. Background Research

1.1. Introduction

Geoscientists implement traditional machine learning techniques on interpretation of seismic sections for nearly 20 years. Although there had been a lot of attempts on different algorithms, the interpretation results were usually not good enough comparing with the results interpreted by human expert in particular for image classification. With the fast development in both GPU computing and computer vision over the last 5 years, more and more geoscientists try to use deep learning instead of machine learning on the same task.

Deep learning techniques, to which Convolutional Neural Networks (CNNs) belong to, can be applied to the image itself, thus accounting for more features. Unlike traditional networks, the CNNs learn to extract the most relevant features by themselves and this characteristic help it stands out from the competition with traditional techniques which usually work with attribute values only. There have been several successful applications of deep learning techniques in fault detection, salt interpretation with seismic sections achieved by other groups.

It usually takes several months for an expert's team to accurately interpret all seismic sections of a field. Perenco, a leading independent Oil & Gas company, which has a lot of field data to interpret, now tries to use both machine learning and deep learning techniques to improve data analytics and fasten the deliverability.

As a part of this ambition, this project is designed to get models applying deep learning techniques especially CNNs, for seismic facies classification (Figure 1).

The field data used in this project comes from a test field in Gabon, Africa. All the SEGY data for this project is owned and provided by Perenco. 5 IL (In-lines) sections are used to train, validate and test the techniques. In total, the test field is covered by 1100 IL (In-lines) and 1400 CL (Cross-lines).

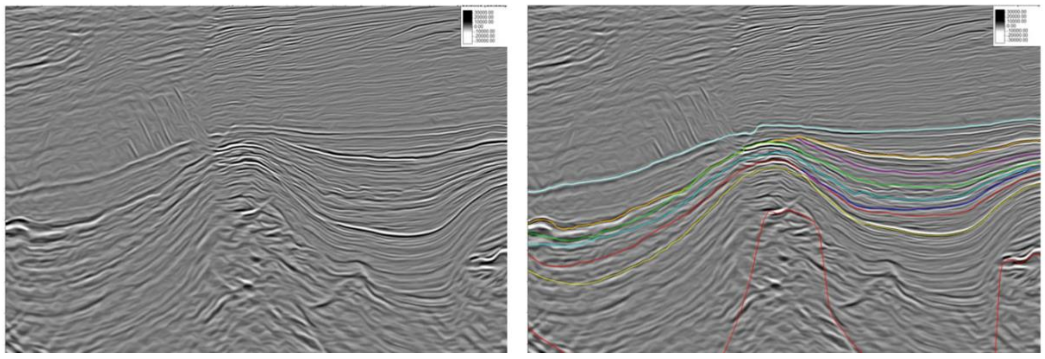


Figure 1. (a) Example seismic section. (b) Same section after interpretation by human.

1.2. Literature Review

Classifying seismic facies in seismic sections is different from general CNNs image identification process. In this case, all characteristic facies are contained in one big image which is usually in dimension thousands by thousands. It's hard to train models separating different facies as compared to models trained on MNIST database, on which, each image represents only a single class.

Two of the most well accepted models on tackling this challenge are mini-patch model and encoder-decoder model. On mini-patch model, one of the well-known groups is led by Waldeland. He shared a basic deep learning methodology to identify salt in seismic sections (Waldeland, 2018). The dataset was a 3D dataset acquired from the Barents Sea. With his focus on salt identification, Waldeland classified each single pixel in the whole datasets as just 'salt' or 'not salt'. Then he considered each pixel individually and selected a 65 * 65 * 65 cube around it as single training object. The corresponding label was the label of each center pixel.

After testing the number of layers, the number of nodes in each layer, pooling, dropout and different activation functions for the network architecture, Waldeland came out with a network configuration (Figure 2) which has 5 convolutional layers and 1 fully connected layer. As claimed in the article, none of the above conditions he had changed affects the results much, so he just implemented batch norm and data augmentation to improve the model accuracy. The model was trained on one manually labeled in-line section (Figure 3a) and the result tested on another in-line section is shown in Figure 3b (Waldeland, 2018). To be noticed, Waldeland only trained it on a 3D seismic cube, for our situation where we deal with 2D seismic sections, the patch should be in size 65 * 65 instead of 65x65x65.

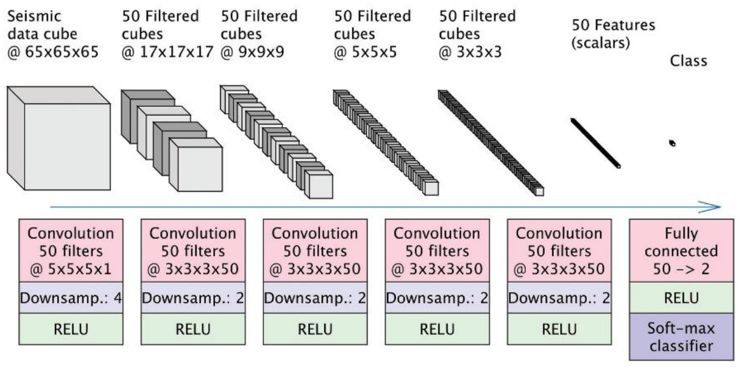

Seismic data cube @ 65x65x65
50 Filtered cubes @ 17x17x17
50 Filtered cubes @ 9x9x9
50 Filtered cubes @ 5x5x5
50 Filtered cubes @ 3x3x3
50 Features (scalars)
Class
Convolution 50 filters @ 5x5x5x1
Downsamp.: 4
RELU
Convolution 50 filters @ 3x3x3x50
Downsamp.: 2
RELU
Convolution 50 filters @ 3x3x3x50
Downsamp.: 2
RELU
Convolution 50 filters @ 3x3x3x50
Downsamp.: 2
RELU
Convolution 50 filters @ 3x3x3x50
Downsamp.: 2
RELU
Fully connected 50 -> 2
RELU
Soft-max classifier

Figure 2. The neural network used for salt picking.

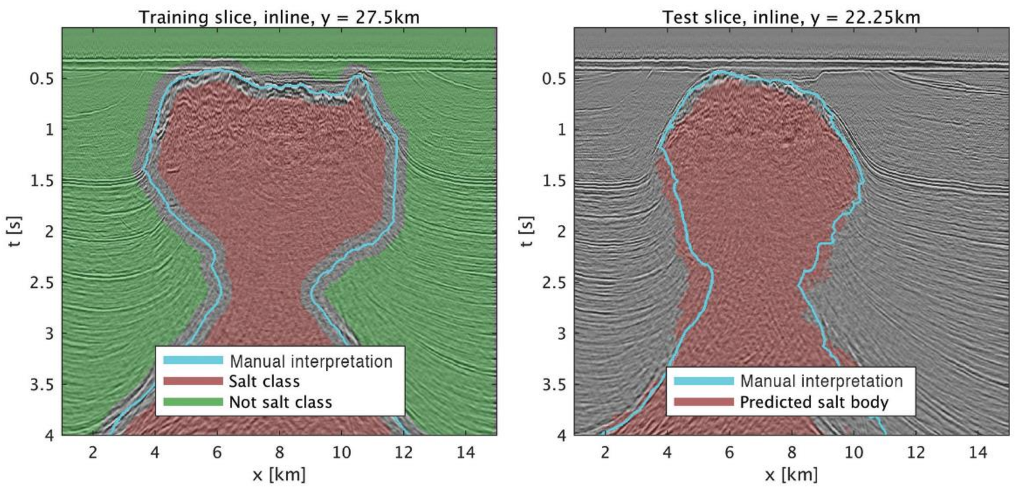


Figure 3. (a) The inline section used for training. (b) The test section.

The methodology of encoder-decoder model I use in this project comes from Ronneberger (Ronneberger, 2015). A network architecture that uses fully convolutional layers as encoder and decoders was designed to segment the biomedical image dataset, which contains different objects in one image as well.

Based on traditional “fully convolutional network” (Long, 2014), Ronneberger modified and extended the architecture of the neural network to a neural network called ‘U-Net’ (Figure 4). In usual encoder and decoders, the high-dimensional input image is first encoded to low-dimension latent vectors and would be reconstructed to high-dimension input image. In ‘U-Net’, with the similar idea of a usual encoder, the input image is first extracted to an image in same dimension as the input image, but much smaller. In the decoder, it not only studies from the extracted image itself, but also the learned feature at the same depth in the encoder (Ronneberger, 2015).

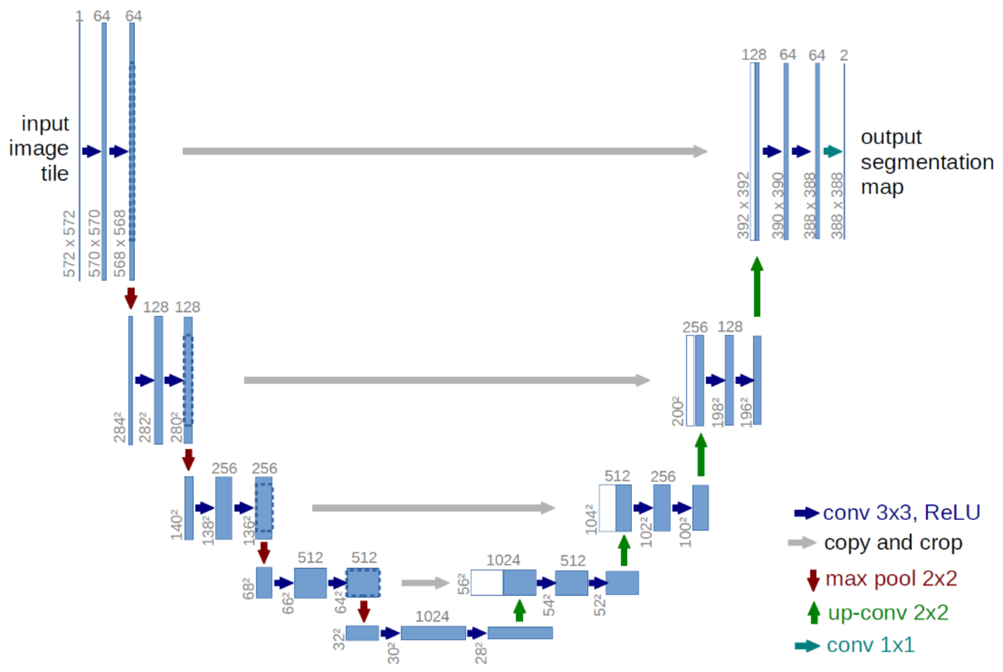


Figure 4. U-Net architecture (example for 32x32 pixels in the lowest resolution).

With similar data type and target, it is rational to try to classify seismic facies and other geological structures using CNNs on the dataset of the test field.

2. Software Description Life Cycle (SDLC)

2.1. Technical Backend and Development Methodology

This project is developed as a standalone code with Python on Jupyter Notebook. All neural network models are implemented using PyTorch, an open source library from Facebook. First developed on RedHat 8 system, this project had been transferred and continued develop on Windows 10 system for GPU support of CUDA 10.0 Toolkit from Nvidia.

The goal of this project is certain and clear, so using a Waterfall methodology is the most appropriate and effective.

2.2. Dataset

2.2.1. Data format

To design the workflow, we need first to understand the dataset. All the data provided by Perenco for this project is in SEGY format. This format is the standard format used for storing geophysical data developed by the Society of Exploration Geophysicists (SEG). These data contain not only the actual field trace data but also many textual explanatory files as shown in Figure 5 (Rune, 2017).

This format of data is not familiar with mainstream programming languages such as python, so an external library is needed to load the data to the Jupyter Notebook.

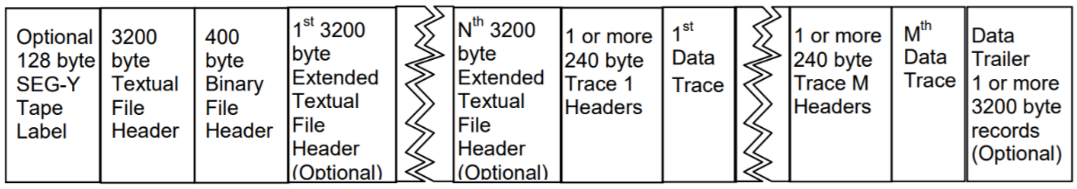

Optional 128 byte SEG-Y Tape Label
3200 byte Textual File Header
400 byte Binary File Header
1st 3200 byte Extended Textual File Header (Optional)
Nth 3200 byte Extended Textual File Header (Optional)
1 or more 240 byte Trace 1 Headers
1st Data Trace
1 or more 240 byte Trace M Headers
Mth Data Trace
Data Trailer 1 or more 3200 byte records (Optional)

Figure 5. Byte stream structure of a SEG-Y file with N Extended Textual File Header records and M traces records.

2.2.2. Dataset Feature

A deep learning model trained with more input data is usually superior to model trained with less input data. The input data added, however, is not necessarily using the same features. New independent features are available from the processed seismic sections of the same in-line which come from a professional geophysical software namely OpendTech®. For that concern, another 5 independent attributes are trained within this project besides the reflectivity. These attributes include cosine phase, dominant frequency, polar dip, spectral decomposition in 20Hz and texture. For convenience, I will use ‘1-feature’ dataset to represent the original seismic section of reflectivity and ‘6-feature’ dataset to represent the original seismic section of reflectivity as well the processed seismic section of other attributes in this report. The models trained with both ‘1-feature’ and ‘6-feature’ datasets are used to measure whether it is worth adding more attributes to improve the results.

For each in-line section, there are 6 seismic images with the reflectivity data and 5 other attributes. Another image data with the same size as other seismic images is essential to indicate the class label of each pixel in the seismic section. For convenience, I will use class label map to represent this data (Figure 6).

2.2.3. Dataset Split

The seismic sections need to be separated into training, validation and test datasets before training the models. This is important as a step to inspect and verificate the models trained. The training dataset is the dataset we use to train the model and test dataset is the unseen dataset we use to evaluate the model. A validation dataset, behaves like a test dataset and is used for frequent evaluation within the training of the model. We use this dataset to optimize the hyperparameters of the model. The seismic sections in each dataset should be selected as independent as possible covering the field to avoid high correlation between each of them. Then, the test dataset is hidden, the models are trained with only training and validation datasets.

2.3. Design Rationale

2.3.1. General Workflow

In this study, four different types of model have been successfully built in total. The first model is trained in traditional feed-forward neural network which can be only trained on ‘6-feature’ dataset with 6 attribute inputs for each pixel. All the following models can be trained on both ‘1-feature’ and ‘6-feature’ datasets. The second model is a CNN model called mini-patch model which slices seismic sections to mini-patches with the same dimension around

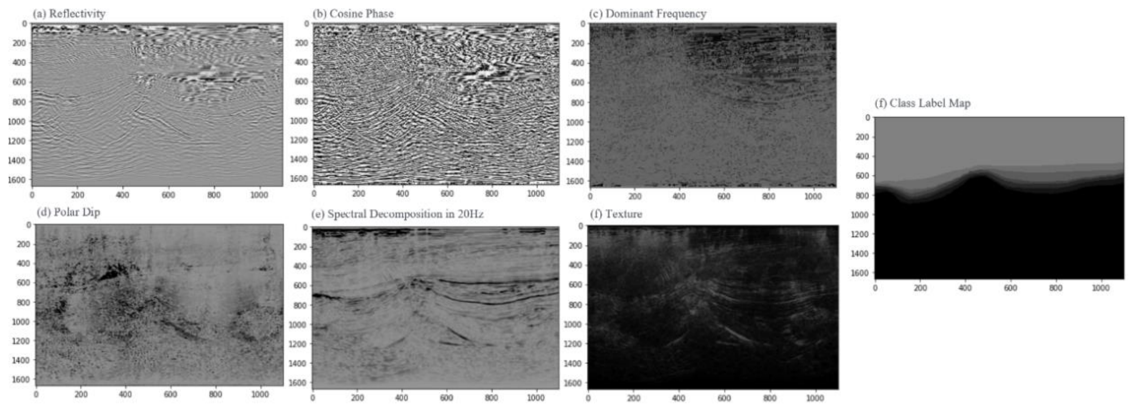


Figure 6. Input features for one in-line

each pixel. The third model is called encoder-decoder model. This CNN model uses whole seismic sections as single training objects using fully convolutional layer-decoder and encoders. The fourth model is a ensemble model combining the results of both the CNN models (the second and the third). This fourth model runs all successfully built CNN models and records the classified result for each pixel as the most frequent label predicted by all those models. This model is the most effective model for seismic facies classification as we will see.

The workflow chart of this project is shown in Figure 7.

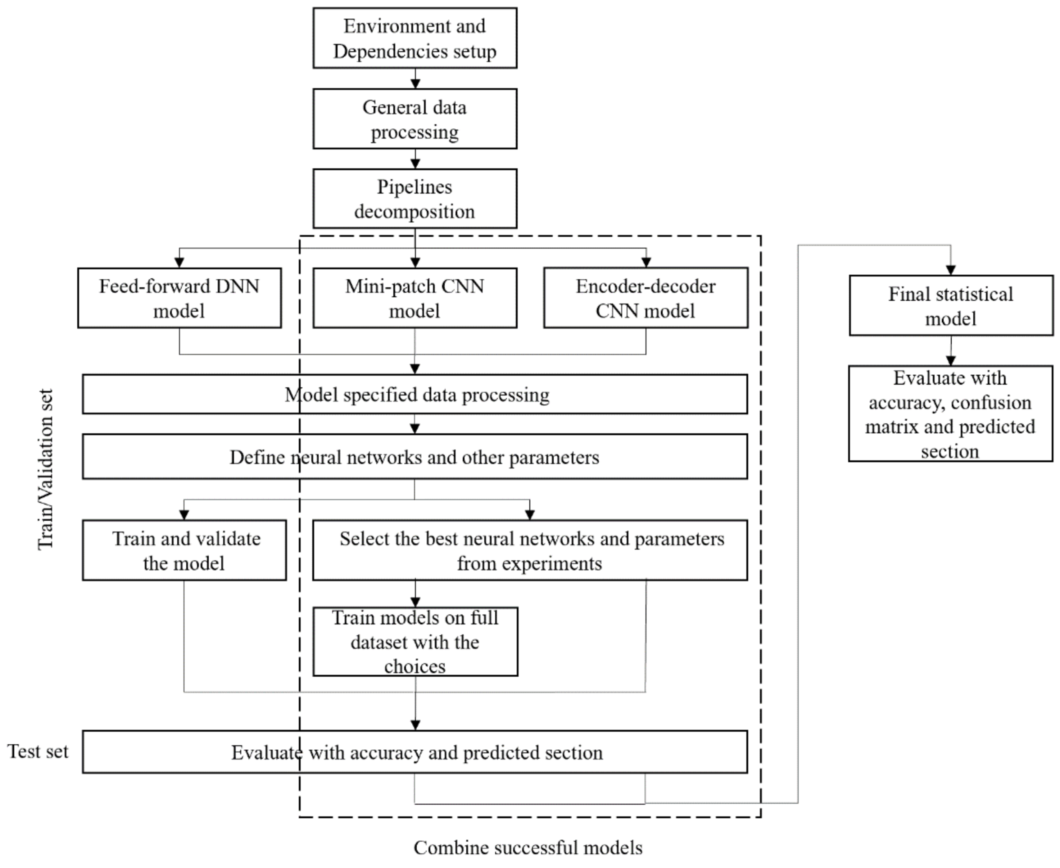

Environment and Dependencies setup
General data processing
Pipelines decomposition
Feed-forward DNN model
Mini-patch CNN model
Encoder-decoder CNN model
Model specified data processing
Define neural networks and other parameters
Train and validate the model
Select the best neural networks and parameters from experiments
Train models on full dataset with the choices
Train/Validation set
Test set
Evaluate with accuracy and predicted section
Final statistical model
Evaluate with accuracy, confusion matrix and predicted section
Combine successful models

Figure 7. Architectural solution diagram

2.3.2. Feed-forward DNN Model

The workflow for the three pipelines is similar but the actual training process after each model is very different.

Feed-forward DNN model is trained based on each single pixel of the seismic section. DNN, representing Deep Neural Network, is used here only for consistency of naming. With '1-feature' dataset, each pixel only has one value corresponding to its reflectivity. Although theoretically feasible, training with only one value as input and applying multiple layers and neurons on it incurs a lot of noises which makes the model unstable. For this reason, this model is only trained on '6-feature' dataset in this project. The 6 attributes, although not much as well, at least provide more information and contribute to make a more stable model.

After some model specified data preprocessing, we need to define the neural network and other parameters. The architecture of the neural network in this model should be simple and straightforward. There is no CNN involved, the layers are fully connected which is easy to design and adjust.

I first train the neural network on the training dataset. Not expecting a good result and not going to optimize much on this model. The metric used to evaluate the model performance was the classification accuracy. After training, the model was evaluated on the test dataset. High accuracy score for test dataset doesn't necessarily prove that the model is good. For example, if there are only two classes of object in the training dataset and the amount of the object in the first class is 10 times that in the second class. The model can always classify a pixel as in the first class and still get an accuracy of over 90%, but which is meaningless. In case of that, besides accuracy, the trained model needs to predict each pixel in a seismic section, and only be evaluated from the predicted output label map.

2.3.3. Mini-patch CNN Model

With more attention on CNN models, both CNN models are better designed and validated. The first CNN model of this project is the mini-patch CNN model. Using not only the pixel itself but also the neighbors around it, this model can be trained on both '1-feature' and '6-feature' datasets.

Different neural networks are tested in this model. 4 networks modified from some mainstream neural networks for image classification and 1 created by me are tested combining with other parameters and at last, the networks have good performance on both training and validation datasets are selected. To get better predicted results, there is not necessarily to use just one best network for this model.

The evaluation process is achieved via 'training versus validation' stage. Each combination of neural network and parameters is set to train a model on training and validation datasets in fixed epochs. The performances of these combinations are then evaluated by not only the accuracy of both training and validation datasets after that but also the f1 score and the trend of the accuracy improvement.

After that, the selected neural network + parameters are trained again. This time, the validation dataset is no longer used for evaluation. It is trained together with the training dataset. We call this a full dataset training.

Same as in the feed-forward DNN model, the trained CNN model needs to be evaluated from both the accuracy and the image domain as well.

2.3.4. Encoder-Decoder CNN Model

The third pipeline corresponds to the encoder-decoder model. Regarding the whole seismic section as a training object, this model can be trained on both '1-feature' and '6-feature' datasets as well.

Same as for last CNN model, different neural networks are tested in this model.

The evaluation process is still achieved via training on training versus validation dataset stage. However, the model isn't necessarily to be trained on full dataset again because this model includes the validation set during the model training stage in design.

The model is evaluated in the same way as in the previous models.

2.3.5. Final Ensemble Model

After the three pipelines, there is an extra pipeline corresponding to the final ensemble model.

Different models are trained in the previous pipelines and they may have different characteristics. For example, a model may be sensitive to a class with horizontal features while another model may be sensitive to another class with vertical features. The idea of this ensemble model is just combining these models together statistically. Combining the predicted results from all successful models belonging to these two CNN models, each pixel in this ensemble model is predicted as a having the class label which has the most occurrence. If two or more different labels have the same votes, the ensemble model will just take the predicted result coming from the member model which has the best accuracy score.

This model, as the ultimate model, is not only evaluated from the accuracy but also on predicted class label map and the confusion matrix.

3. Implementation

3.1. Description and Library Dependencies

The code of this project can be accessed through https://github.com/msc-acse/acse-9-independent-research-project-czp21c.

All the open source libraries I have used in this project is shown in Table 1:

Library Name	Dependency	Description
numpy	high	save data.
torch	high	train deep neural networks.
livelossplot	medium	plot accuracy and loss in model training process.
obspy	medium	load SEGY data.
torchvision	medium	transform the input image.
scikit-learn	medium	provide machine learning metrics.
matplotlib	medium	plot the data.
os	low	check the files on storage.
pycm	low	plot the confusion matrix.
scipy	low	calculate the mode.
resource	low	set the resource limit.
torchsummary	low	summary the neural network.
math	low	provide some math functions.

Table 1. Open source libraries used in this project.

3.2. General Data Preprocessing

To load the SEGY data in Python, a library ObsPy (Krischer, 2015) has been used. This library can read the information of SEGY data automatically and which data after ObsPy's operation can be exported to numpy arrays easily. Before loading the data, we need to define how many classes do the seismic sections contain and create the class label map for each seismic section. There are 7 different classes in total, which corresponds to 7 geological formations. Geological formations are usually separated according to the lithology or geological time and this information is usually well reflected on seismic sections. All SEGY files are originated from a professional geophysical software namely OpendTech® and the original size of one seismic section is 1667 * 1101.

After loading all the SEGY files in Python, I get three arrays stored by numpy. The first two arrays are my training arrays. The first array contains only reflectivity data for all the 5 seismic sections with shape (5, 1667, 1101), where 5 stands for the number of seismic sections, 1667 stands for the width of a seismic section and 1101 stands for the length of a seismic section. The second array contains all the attributes data with shape (5, 6, 1667, 1101) where 5 stands for the number of seismic sections as well, 6 stands for the number of attributes. 1667 and 1101 still stand for the width and of a seismic section. The third array is my target array which is the class label map whose shape is (5, 1667, 1101) same as the first training array.

The dataset then needs to be separated into training, validation and test datasets. With only 5 seismic sections, I use the standard 3:1:1 to split the original datasets which means 3 whole seismic sections as training datasets and 1 different section for each validation and test dataset. It is noticeable that although the training may happen on single pixels or mini patches around pixels during the training process in some models, we should separate the datasets by whole In-line section. For convenience, I always use In-line 3223, In-line 3373 and In-line 3473 as training datasets for all the models. The validation dataset is In-line 3293 and the test dataset is In-line 3424. The models trained using this standard are then good to compare.

3.3. Test Section

The test dataset is selected to be *In-line 3424*. This section is shown in Figure 8 with all its 6 attributes in python.

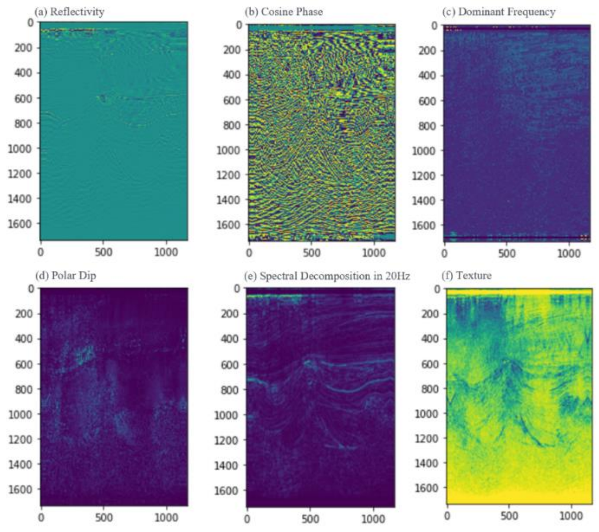


Figure 8. Features for Test *In-line 3424*.

And the corresponding label map for *In-line 3424* is shown in Figure 9

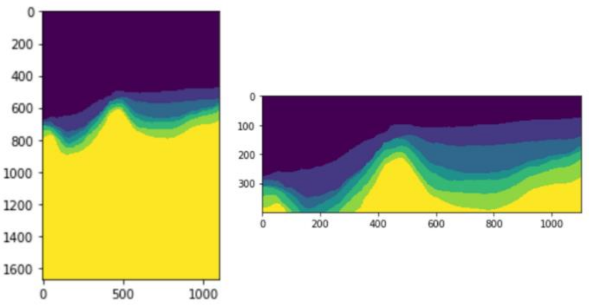


Figure 9. (a) Label map for test *In-line 3424*. (b) Central part of test *In-line 3424*.

3.4. Feed-Forward DNN Model

3.4.1. Data Compiling

The datasets then need to be preprocessed with some model specified requirements.

Usually, the training objects we feed in the neural networks are designed to contain the image or attribute information itself. For example, the training array for MNIST database has shape (50000, 28, 28) which has 3.92 * 10^7 numbers. The training array in this model with shape (5, 6, 1667, 1101) has 5.51 * 10^7 numbers which seems OK. However, when dealing with mini-patch CNN model which is trained after, if we store each mini-patch as a single training object, it will have 2.33 * 10^{11} numbers in total while having lots of repeating data, which will impose huge burden on the memory to store those data. To deal with this, I only record the coordinate (3 numbers) of each pixel. The attributes will be loaded in the PyTorch dataset which will be introduced later.

The training coordinates needs to be balanced in the feed-forward DNN model. We need to make sure that each class has basically the same number of training objects before training the model or the predicted results from the trained model will be heavily imbalanced. Data preprocessing is needed to avoid that problem at the beginning. I solve this problem by using the coordinates created before for memory saving. A benchmark needs to be set for each of the classes beforehand. When a class has more training objects than the benchmark, it will randomly abandon some of the training objects until the number of them gets close to the benchmark. Relatively, a class that does not have enough training objects will randomly append new training objects using data augmentation: new created objects will be the duplication of existing objects in the same class but with some slight modifications. The actual data augmentation algorithms I used will be covered later. At first, I sort the coordinates by their corresponding label. There are 7 different classes in this project. Not surprisingly, the first and the last class have times of training objects more than the other classes. A lot of training objects belong to these two classes are far away from objects with other labels. To avoid losing structural information of the whole section because of the random deletion, I create two arrays to store some characteristic objects in the first and last class. At the end of the balancing, these characteristic coordinates will be added to the training coordinates manually. This balancing process would take a lot of time, so I save the balanced training coordinates and the labels as *npy* files to avoid creating new coordinates every time in actual training process.

The data augmentation algorithms I implemented in this model includes both prior data augmentation and dynamic data augmentation. As another important step in the data preprocessing, data augmentation increases the training dataset by slightly changing the existing training objects. It is a good method against overfitting. Basically, there are two types of data augmentation theory: prior data augmentation and the dynamic data augmentation. Prior data augmentation adds more training data at the beginning. Like manually adding more training objects, the model is benefited with the bigger training dataset. In the prior data augmentation of feed-forward DNN model, each attribute of a training pixel changes within a small range according to a moduler corresponding to a hundredth of the maximum value of that attribute. Because the value distribution of each attribute is differ greatly, the augmentated data has no chance to change to another class. In dynamic data augmentation, the size of the training dataset is unchanged, however, in this time, each training object has a chance to be changed in some way during each training epoch. The model has new objects to train in each epoch. Not training on exact same dataset each time, the overfitting problem can be effectively control. In the dynamic data augmentation of this model, each attribute in each pixel has a small probability to change randomly within the same range as in prior data augmentation in each epoch. The probability I have used in this model is 10%.

For validation and test datasets, I only store the coordinates and the corresponding labels and don't apply balancing on them.

All datasets then need to be normalized according to the mean and standard deviation of each attribute.

To train model through PyTorch, we need first transform the data to PyTorch Tensors. After that, custom PyTorch datasets are needed to return the actual attribute data from the coordinates. I create three different datasets for different purposes. The first dataset is called *ValidationDataset_ff* which just takes the coordinates and returns the attributes of the pixels with the labels according to them. The second dataset is named *TrainDataset_original_ff* which dataset not only returns the corresponding labels of the pixels with the attributes but also does prior data augmentation. The third dataset is named *TrainDataset_dynamic_ff*. This custom dataset is designed for dynamic data augmentation.

The training dataset and training coordinates have then been loaded by the *TrainDataset_original_ff* together while the validation and test datasets have been loaded by the *ValidationDataset_ff* with their according coordinates at first.

3.4.2. Neural Network

After loading the datasets to the PyTorch Dataloader, the neural network needs to be set. Because each pixel, as my design, only has 6 different attributes, the architecture of the neural network is simple as shown in Figure 10.

3.4.3. Training

The weights in the neural network are updated during the training on the basis of the mismatch measured by the loss function. Model gets trained and optimized by trying to minimize that loss. There are lot of different loss criterions. As a double insurance for the imbalanced dataset, focal loss (Lin, 2017) is used in this model as the loss criterion. Focal loss is designed specially to cope with imbalanced data in different classes during training. The loss function of cross entropy can be simplified as $CE(p, y) = -\log(p_t)$ where:

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{otherwise,} \end{cases}$$

In the above, $y \in \{\pm 1\}$ corresponds the class and $p \in [0,1]$ is the model's estimated probability for the class with label $y = 1$.

Based on this expression, the loss function of focal loss is:

$$\mathrm{FL}(p_t) = -(1 - p_t)^{\gamma} \log(p_t).$$

The added part $-(1-p_t)^{\gamma}$ is named modulating factor. When an example is misclassified and p_t is small, the modulating factor is close to 1 and the loss is unaffected, when p_t close to 1, the factor goes to 0 and the loss for well-classified examples is down weighted. The focusing parameter γ controls the down weighted rate of easy examples (Lin, 2017). The code of focal loss comes from https://github.com/clcarwin/focal_loss_pytorch. I set γ to 2 in this project as suggested by the paper.

The optimizer I used in this project is Adam optimizer (Kingma, 2015). The learning rate I used for this model is 0.01 while the weight decay is set to be 0.

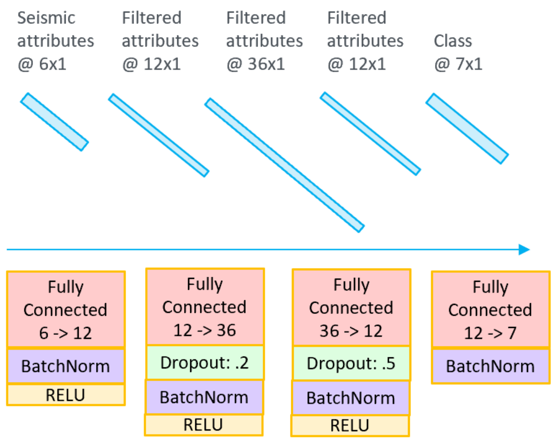


Figure 10. Feed-Forward net architecture.

3.4.4. Result

The model is trained properly. However, the learning process seems extremely slow and never improves on validation dataset during 40 epochs. The loss and accuracy of both training and validation datasets fluctuate around some fixed values. The final training accuracy is 35.2% while that of validation is 58.7%.

Using the width of a seismic section as the test batch size in test loader, the predicted class label map on test section can be easily stored and plotted (Figure 11).

The final accuracy of feed-forward DNN model using '6-feature' dataset on the test seismic section is about 9.0% which is extremely low.

3.4.5. Conclusion

The performance of this model is bad on both the accuracy and the predicted class label map on test section. Because this project is not design for testing traditional machine learning algorithms or optimizing feed-forward models, I didn't test different hyperparameters or other networks for this model. The result coming from this model will only be considered as a reference to the afterwards CNN models and will never be used in the ensemble model.

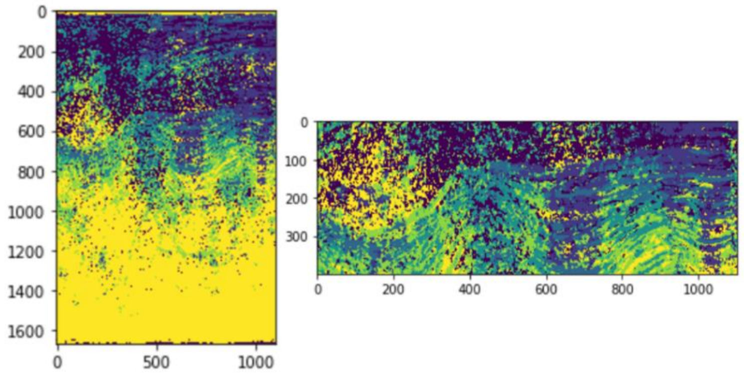


Figure 11. The predicted class label map on test section for feed-forward DNN Model.

3.5. Mini-patch CNN Model

3.5.1. Data Compiling

The data preprocessing steps for mini-patch CNN model is similar to those in the feed-forward DNN model.

For mini-patch CNN model, at first, we need to define a mini patch size for each training patch. I use 65 in this project as recommended by Waldeland which indicates 32 neighbors up and down, left and right. To make sure the training works good near the edges, mirror padding has been implemented on the original seismic sections. The padding size is 32 for this project so the training arrays are (5, 1731, 1165) and (5, 6, 1731, 1165) after padding.

All the coordinates are created in the same way as in the feed-forward DNN model. We can use the *npy* file created before to save some time.

All three datasets then need to be normalized according to the mean and standard deviation of each attribute.

This CNN model can be trained both with '1-feature' and with '6-feature' datasets. For convenience, I will use '1-feature' mini-patch model and '6-feature' mini-patch model to represent them.

For '1-feature' mini-patch model, the first task is to construct custom datasets same as for feed-forward model. For same reason, there are three different datasets as well which are now named *ValidationDataset*, *TrainDataset_original* and *TrainDataset _dynamic*. Different than the datasets for feed-forward neural networks, the datasets read small patches of image not pixels at this time.

The custom datasets for '6-feature' mini-patch model is a little bit different than the '1-feature' version. The PyTorch datasets, with names: *ValidationDataset_3D*, *TrainDataset _original_3D* and *TrainDataset_dynamic_3D* get 4 indexes to parse now while the added one represents the dimension of different attributes. The datasets read small blocks not images at this time.

With the same idea as in the feed-forward DNN model, data augmentation is necessary in this model as well. For this model, I use both prior data augmentation and dynamic data augmentation as well. Data augmentation for small patches and blocks is a little complex. Common data augmentation libraries, for example the *transforms* from *torchvision* in PyTorch supports only to RGB image, which means each pixel of the input image must has an integer value between 0 and 255. The transform functions take the values and return a transformed image which has array with values from 0 to 1. However, the datasets cannot be transformed through this way directly. The value in each pixel is seismic attributes not RGB value which range from a single digit to several thousands. To deal with that, in both types of data augmentation, the candidate patch needs to be transformed to a patch with all its value integers between 0 to 255 manually at first.

For '1-feature' model, in prior data augmentation, the patch is designed to rotate within 15 degrees, translate between 0.05 up and down, left and right and scale between 0.95 and 1.05 which can be executed by RandomAffine function from torchvision.transforms. For dynamic data augmentation, each small patch in each epoch of training will have a small probability (20% in this model) to be transformed with the same type of transformation as in prior data augmentation.

Data augmentation for '6-feature' model is a little different than in the '1-feature' model. In prior data augmentation, the block with shape (6, 65, 65) will first be split into patches with size (65, 65) each and then be transformed with the same way as in '1-feature' model separately. These transformed patches will be stacked together at last as the new block. In dynamic data augmentation, each block in each epoch of training will has 20% chance to change as well.

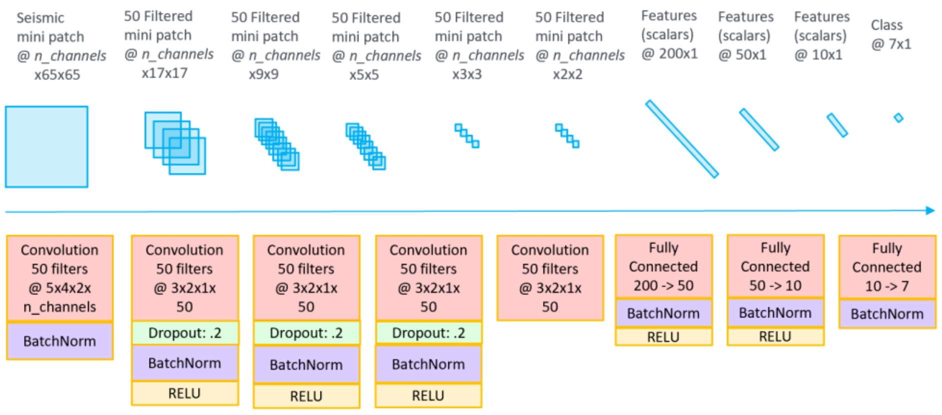
3.5.2. Neural Networks

After loading the datasets to the PyTorch Dataloader, the neural networks need to be set. As introduced before, the model can be trained on both '1-feature' dataset and '6-feature' datasets. For the models trained on different datasets, however, even if the shape of them is different, they can share a same series of neural networks. Including the input channels as an argument, the network can be used flexibly. I have used 5 different networks for this model.

The first net I use is like a 2D version of the net shown in Figure 2 which has been named *BasicNet* (Figure 12). This network comes from a Github Repository: https://github.com/LukasMosser/asi-pytorch created by Lukas Mosser which has 102,671 parameters for input size 65 * 65. The second network is created by myself. This network combines the first net and some idea of *AlexNet* (Krizhevsky, 2012) which has been named *OptimizedNet* shown in Figure 13. With different filter size in each layer and a Maxpooling layer, the network has 755,357 parameters for the same input size. Other three networks all come from the ResNet family. I implement *ResNet 18, ResNet 34 and ResNet 50* in this project (Figure 14). The code for ResNet are modified from the official PyTorch ResNet (https://pytorch.org/docs/stable/_modules/torchvision/models/resnet.html). ResNet optimizes the training of networks by using residual learning framework. The differences in each layer of models in Figure 14 correspond to different type and number of residual block which have been used. *ResNet*s uses a lot of parameters. *ResNet 34*, for example, for input size 65 * 65, uses 21,281,991 parameters which is nearly 34 times more than the *OptimizedNet*.

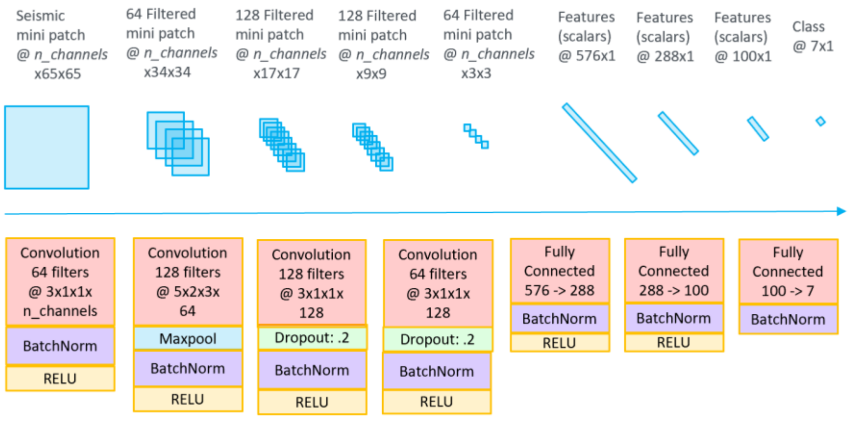
3.5.3. Training and Validation

Using Adam optimizer and Focal loss, the models trained with mini patch CNNs are comparable with the feed-forward DNN model. GPU support is needed for this model. Because data privacy is very important for companies, all data I get from Perenco are not allowed to upload to Google Colaboratory for training. In case of that, I have installed CUDA 10.0 Toolkit on a Windows machine, and with support from an Intel® graphic card, the model can be trained on regular Jupyter Notebooks.



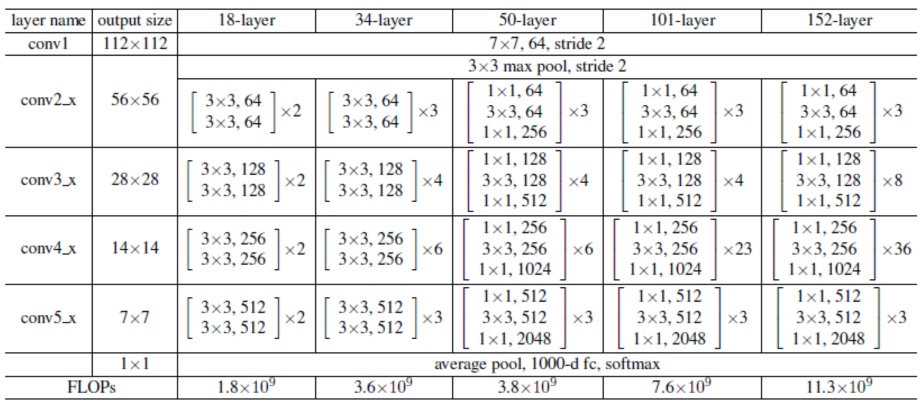
Seismic mini patch @ n_channels x65x65
50 Filtered mini patch @ n_channels x17x17
50 Filtered mini patch @ n_channels x9x9
50 Filtered mini patch @ n_channels x5x5
50 Filtered mini patch @ n_channels x3x3
50 Filtered mini patch @ n_channels x2x2
Features (scalars) @ 200x1
Features (scalars) @ 50x1
Features (scalars) @ 10x1
Class @ 7x1
Convolution 50 filters @ 5x4x2x n_channels
BatchNorm
Convolution 50 filters @ 3x2x1x 50
Dropout: .2
BatchNorm
RELU
Convolution 50 filters @ 3x2x1x 50
Dropout: .2
BatchNorm
RELU
Convolution 50 filters @ 3x2x1x 50
Dropout: .2
BatchNorm
RELU
Convolution 50 filters @ 3x2x1x 50
Fully Connected 200 -> 50
BatchNorm
RELU
Fully Connected 50 -> 10
BatchNorm
RELU
Fully Connected 10 -> 7
BatchNorm

Figure 12. The *BasicNet* architecture.



Seismic mini patch @ n_channels x65x65
64 Filtered mini patch @ n_channels x34x34
128 Filtered mini patch @ n_channels x17x17
128 Filtered mini patch @ n_channels x9x9
64 Filtered mini patch @ n_channels x3x3
Features (scalars) @ 576x1
Features (scalars) @ 288x1
Features (scalars) @ 100x1
Class @ 7x1
Convolution 64 filters @ 3x1x1x n_channels
BatchNorm
RELU
Convolution 128 filters @ 5x2x3x 64
Maxpool
BatchNorm
RELU
Convolution 128 filters @ 3x1x1x 128
Dropout: .2
BatchNorm
RELU
Convolution 64 filters @ 3x1x1x 128
Dropout: .2
BatchNorm
RELU
Fully Connected 576 -> 288
BatchNorm
RELU
Fully Connected 288 -> 100
BatchNorm
RELU
Fully Connected 100 -> 7
BatchNorm

Figure 13. The *OptimizedNet* architecture.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\left[\begin{array}{c}3\times3, 64\\3\times3, 64\end{array}\right]\times2$	$\left[\begin{array}{c}3\times3, 64\\3\times3, 64\end{array}\right]\times3$	$\left[\begin{array}{c}1\times1, 64\\3\times3, 64\\1\times1, 256\end{array}\right]\times3$	$\left[\begin{array}{c}1\times1, 64\\3\times3, 64\\1\times1, 256\end{array}\right]\times3$	$\left[\begin{array}{c}1\times1, 64\\3\times3, 64\\1\times1, 256\end{array}\right]\times3$
conv3_x	28×28	$\left[\begin{array}{c}3\times3, 128\\3\times3, 128\end{array}\right]\times2$	$\left[\begin{array}{c}3\times3, 128\\3\times3, 128\end{array}\right]\times4$	$\left[\begin{array}{c}1\times1, 128\\3\times3, 128\\1\times1, 512\end{array}\right]\times4$	$\left[\begin{array}{c}1\times1, 128\\3\times3, 128\\1\times1, 512\end{array}\right]\times4$	$\left[\begin{array}{c}1\times1, 128\\3\times3, 128\\1\times1, 512\end{array}\right]\times8$
conv4_x	14×14	$\left[\begin{array}{c}3\times3, 256\\3\times3, 256\end{array}\right]\times2$	$\left[\begin{array}{c}3\times3, 256\\3\times3, 256\end{array}\right]\times6$	$\left[\begin{array}{c}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{array}\right]\times6$	$\left[\begin{array}{c}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{array}\right]\times23$	$\left[\begin{array}{c}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{array}\right]\times36$
conv5_x	7×7	$\left[\begin{array}{c}3\times3, 512\\3\times3, 512\end{array}\right]\times2$	$\left[\begin{array}{c}3\times3, 512\\3\times3, 512\end{array}\right]\times3$	$\left[\begin{array}{c}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{array}\right]\times3$	$\left[\begin{array}{c}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{array}\right]\times3$	$\left[\begin{array}{c}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{array}\right]\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 14. The *ResNet* architectures.

To get good models for mini-patch CNN models, experiments are needed to help us find the best data augmentation algorithm, the best network architecture and the best hyperparameters accordingly.

Basically, the model performs better with more training data. In the *get_effective _data* function, the number of objects in each class after augmentation is defined to be in a certain ratio of that in the class having the least objects. To balance the model performance and the memory use, I test the ratio with 1, 1.5 and 2. The average accuracy is about 2.5% better for each 0.5 ratio increase. It will reach a bottleneck later, but the phenomenon conforms to what I expected.

The experiments for network architecture and hyperparameters are conducted based on the augmented training dataset with ratio 1.5.

Among all hyperparameters, the learning rate is tested to be the most relevant one to the model performance. Although I add scheduler during the training, the model with a better initial learning rate behaves more stable and efficient. So, I use 4 different learning rates with the 5 different networks to determine good networks and a good learning rate.

The learning rates I have tested include 2*10^{-2}, 1*10^{-2}, 5*10^{-3} and 1*10^{-3}, the training and validation accuracies of the models trained on ‘1-feature’ dataset with these combinations within 10 epochs are shown in Table 2:

	2*10^{-2}		1*10^{-2}		5*10^{-3}		1*10^{-3}	
	Training	Validation	Training	Validation	Training	Validation	Training	Validation
BasicNet	88.9%	88.6%	91.1%	88.3%	92.0%	89.8%	92.5%	89.9%
OptimizedNet	97.6%	92.3%	98.4%	92.4%	98.8%	92.4%	99.0%	92.4%
ResNet 18	98.7%	91.6%	99.3%	91.1%	99.5%	91.0%	99.7%	91.3%
ResNet 34	98.6%	91.8%	99.3%	91.9%	99.5%	91.6%	99.7%	91.7%
ResNet 50	98.3%	90.8%	99,1%	90.4%	99.4%	90.5%	99.7%	90.8%

Table 2. Training and Validation accuracies with different parameter combinations on ‘1-feature’ dataset within 10 epochs.

Same experiment happens on ‘6-feature’ dataset and the result is shown in Table 3:

	2*10^{-2}		1*10^{-2}		5*10^{-3}		1*10^{-3}	
	Training	Validation	Training	Validation	Training	Validation	Training	Validation
BasicNet	88.5%	90.5%	90.9%	90.9%	92.0%	91.2%	92.5%	91.1%
OptimizedNet	97.1%	92.4%	98.2%	92.8%	98.6%	93.1%	98.8%	93.0%
ResNet 18	98.4%	92.2%	99.2%	92.0%	99.5%	92.2%	99.7%	92.5%
ResNet 34	98.2%	92.2%	99.1%	92.1%	99.5%	92.1%	99.7%	92.3%
ResNet 50	98.1%	91.8%	99.1%	91.8%	99.5%	92.0%	99.7%	92.0%

Table 3. Training and Validation accuracies with different parameter combinations on ‘6-feature’ dataset within 10 epochs.

The experiment results of ‘1-feature’ and for ‘6-feature’ dataset are similar. A good network should have similar training and validation curves and both good on accuracy. Within 10 training epochs, *BasicNet* performs bad compared with other networks and is eliminated first. *OptimizedNet* easily wins on both ‘1-feature’ and ‘6-feature’ dataset with good accuracies and less differences between the training and validation curves. This network should be selected absolutely. Although facing huge overfitting problems, however, *ResNet*s indicate their fast learning speed on training dataset. Among the *ResNet*s, *ResNet 34* is the best from the results. With a thinking that a larger dataset would relieve overfitting, I select *ResNet 34* as well. With smaller learning rate, the networks have better training accuracies, however there is not a big difference on validation datasets. $1*10^{-2}$ is selected as the best learning rate because it has the best ability to against overfitting among these options.

So I use *OptimizedNet* with learning rate 1e-2 and *ResNet 34* with the same learning rate to train the models on full dataset. The trained models then, are regared as my final mini-patch CNN models.

3.5.4. Results

To train with the full dataset of both '1-feature' and '6-feature' datasets, the models using both options predict well on test section (Figure 15, 16, 17, 18). The model with *OptimizedNet* gets 99.4 % training accuracy while the model with *ResNet 34* gets 99.7 % on the '1-feature' dataset. On '6-feature' dataset, the model with *OptimizedNet* gets 99.3 % training accuracy while the model with *ResNet 34* gets 99.7 %.

The model with *OptimizedNet* gets 96.9 % test accuracy while the model with *ResNet 34* gets 95.2 % on the '1-feature' dataset. On '6-feature' dataset, the model with *OptimizedNet* gets 97.7 % test accuracy while the model with *ResNet 34* gets 96.7 %.

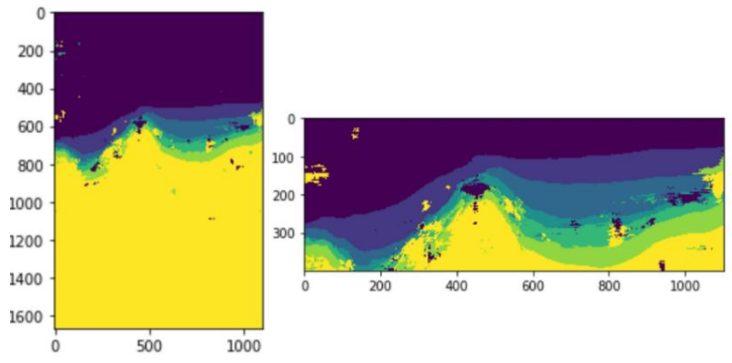


Figure 15. The predicted class label map on test section for mini-patch CNN model using '1-feature' dataset and *OptimizedNet*.

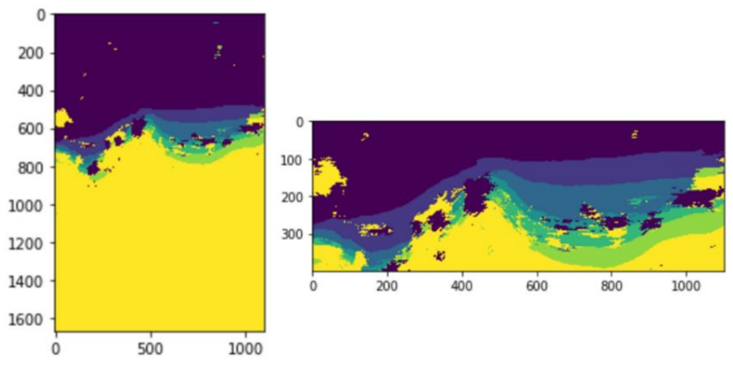


Figure 16. The predicted class label map on test section for mini-patch CNN model using '1-feature' dataset and *Resnet 34*.

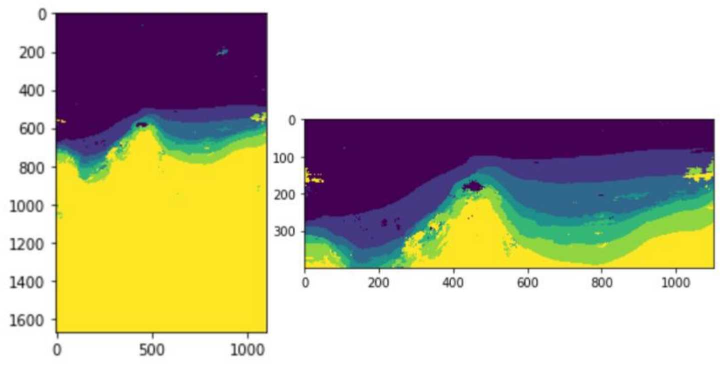


Figure 17. The predicted class label map on test section for mini-patch CNN model using ‘6-feature’ dataset and *OptimizedNet*.

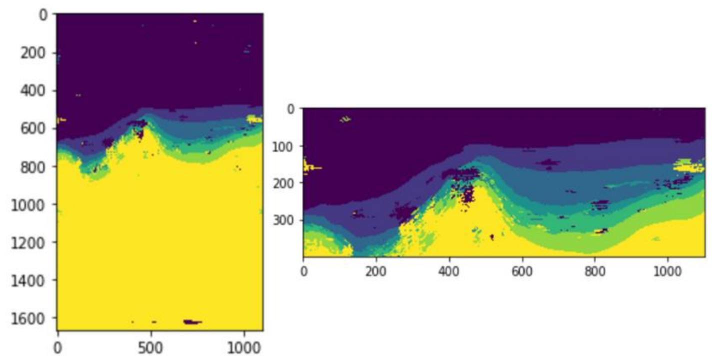


Figure 18. The predicted class label map on test section for mini-patch CNN model using ‘6-feature’ dataset and *Resnet 34*.

3.5.5. Conclusion

The performance of this model is good comparing with that of the feed-forward DNN model on both the accuracy and the predicted class label map on test section. Although the accuracies of these trained models in this section are considered high, the predicted class label maps not quite conform to the actual one. The predicted class label maps lack the ability to indicate geological continuity which is important in geosciences.

3.6. Encoder-Decoder Model

3.6.1. Data Compiling

The core of this model is modified from *pytorch_resnet18_unet.ipynb* in a GitHub repository: https://github.com/usuyama/pytorch-unet. When train with encoder-decoder model, the seismic sections do not need to be sliced or read as mini patches. This time, the seismic sections need to be carefully resized and the new size is strongly related to the U-Net architecture.

In the GitHub repository, author uses the Convolutional neural networks (CNNs) to encode images of size 192x192 to size 6x6 and use up-sampling and concatenation to decode the extracted feature to original size. There are 8 layers of convolutional neural networks in the original architecture for the encoder and 7 layers for the decoder.

The original size of a seismic section for each attribute is 1667x1101 which is nearly 40 times of the example image. Encoding the original seismic section to size about 52x34 is not good enough. To get a better model, I deepen the original network. For that reason, the original seismic sections need to be tailored first. Keep only the center of the sections, the tailored sections have size 1536x1024 for each attribute, by using which size, the original section can be properly extracted by simply adding layers to the neural network. And with this size, the features can be extracted with 4 more layers.

The three arrays now have shape (5, 1536, 1024), (5, 6, 1536, 1024) and (5, 1536, 1024) at this stage.

With only 3 seismic sections for training and seven different classes to separate, back propagating with labels 0-6 is a little hard. To optimize this process, I raise the dimension of each label in the class label map. The class maps before were straightforward. A point in this map may have a value 0, 1, 2… etc. We usually use cross entropy loss for multiple classes as the criterion to optimize the model. However, with very limited data, it may face many unexpected problems and hard to be improved. With that concern, I raise the dimension of the label map. For example, a pixel in class 0 is corresponding to [1, 0, 0, …] now. By doing which, we can now use binary cross entropy not multi-class cross entropy as the criterion which is easier to converge in this situation. After the transformation, the label data now has a shape (5, 7, 1536, 1024). Normalization should be applied on the tailored datasets as well.

To save effort, the '1-feature' dataset which originally had shape (5, 1536, 1024) are transformed to arrays with the shape as '6-feature' dataset before training. With the same dimension, both kinds of data can now be trained using the same series of neural network.

Only one PyTorch custom dataset is needed in this model. This dataset reads the whole seismic section and the corresponding label map as the data and targets. Data augmentation is not ideal in this situation. Because only three seismic sections will be fed for training, adding data augmentation carelessly, for example, rotation or amplitude modification, will make the model unstable. I keep the function to apply data transformation in this custom dataset, but I don't use it for my models.

3.6.2. Neural Network

In the original ResNetUNet architecture as used by the author, the last 7 layers in the encoder are inherited from the official PyTorch ResNet (https://pytorch.org/docs/stable/_modules/torchvision/models/resnet.html).

Considering a residual block layer, either with BasicBlocks or BottleNecks performs better than a simple Convolutional layer, I deepened the architecture by adding 2 more residual block layers after experiments. With less layers, the features cannot be learned properly and with more layers, the model does not improve much let along with more parameters to determine. The blocks in each layer added have the same type as the blocks in the original 4 ResNet block layers. In my modified network, the encoder has 10 layers and the decoder has 9 layers. The network now can extract features small to 12 * 8. After my modification, the new U-Net with ResNet architecture so called *ResNetUNet* is shown in Figure 19.

3.6.3. Training and Validation

I use the combination of binary cross entropy and dice loss (Milletari, 2016) as the criterion for this model. I have tried focal loss inherited from my previous models. However, when using focal loss, the error becomes extremely huge and hard to be controlled or get improved because it calculates the error for the whole seismic section.

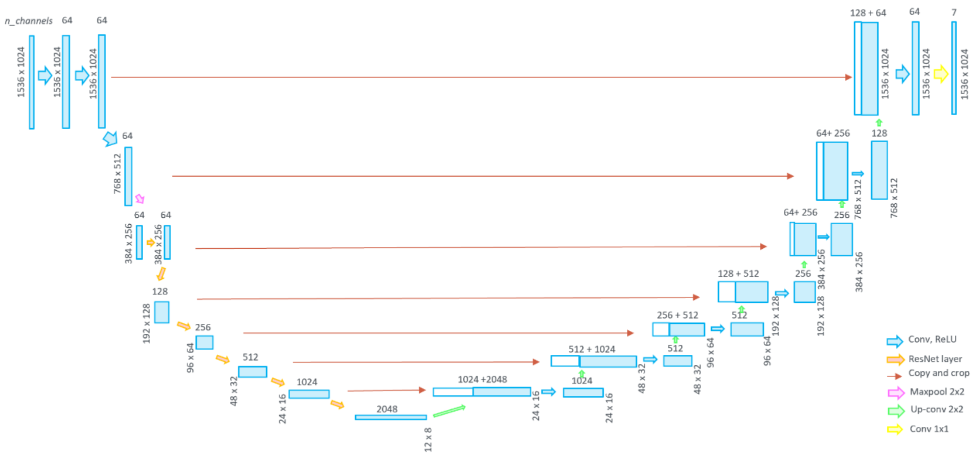


Figure 19. The *ResNetUNet* Architecture.

To deal with this problem, I use the combination of binary cross entropy loss and dice loss in this model as suggested by the GitHub repository. Dice loss is a measure of overlap widely used to assess segmentation performance. Let R be the reference foreground segmentation with voxel values r_n, and P the predicted probabilistic map for the foreground label over N image elements p_n, with the background class probability being 1-P, the 2-class variant of the dice loss can be expressed as:

$$\mathrm{DL}_2 = 1 - \frac{\sum_{n=1}^{N} p_n r_n + \epsilon}{\sum_{n=1}^{N} p_n + r_n + \epsilon} - \frac{\sum_{n=1}^{N} (1-p_n)(1-r_n) + \epsilon}{\sum_{n=1}^{N} 2 - p_n - r_n + \epsilon}$$

where the ε (smooth) is used to ensure the stability (Sudre, 2017). The weights of both loss functions are 0.5 in my model.

The training of this model is very similar for '1-feature' dataset and '6-feature' dataset. The only difference is that there are 6 different sections, not 1, in the second dimension of '6-feature' dataset.

Using Adam optimizer, the models trained with encoder-decoder are still comparable with previous models in some way.

Models trained with encoder-decoder are not compatible with GPU in this project. Because the seismic section is too large, it takes larger than 15 GB memory (the GPU memory of my machine) to save the weights of each model. The models are trained on CPU instead.

Using only BasicBlocks as the residual block for this model, with 6 ResNet block layers in the encoder, I have created and validated 3 different networks with different number of BasicBlocks in the residual block layers. All the networks are modified from the original *ResNet*s. *ResNetUNet 26*, modified from *ResNet 18*, uses 2 BasicBlocks in its all 6 layers. *ResNetUNet 42* and *ResNetUNet 54* are both modified from the *ResNet 34*. *ResNetUNet 42* adds a layer containing 2 BasicBlocks both before and after the original layers with 3, 4, 6 and 3 BasicBlocks respectively. *ResNetUNet 54*, implementing a symmetry thinking, has 3, 4, 6, 6, 4 and 3 BasicBlocks in the layers respectively.

The training process of encoder-decoder CNN models in this project, from the experiments, contains a lot of randomness. The training usually goes into the direction to optimize the binary cross entropy loss fast and only slowly optimize the dice loss.

The best neural network for both '1-feature' and '6-feature' datasets is the *ResNetUNet 54*. This network has the best chance to overcome the local minimum and goes to the correct direction to minimize both binary cross entropy loss and dice loss at the same time.

Then, I train with this architecture on both '1-feature' and '6-feature' datasets and select the models with the least validation loss during the 80 training epochs as my final encoder-decoder CNN models.

3.6.4. Results

The predicted class label maps of encoder-decoder CNN models are superior in indicating the geological continuity. The predicted results from model trained with *ResNetUNet 54* using '1-feature' dataset and '6-feature' datasets on test section are shown in Figure 20 and Figure 21. To be noticed, the test section in this model has been tailored as well, so the predicted class label maps are smaller than those in other models.

The accuracy of the encoder-decoder model trained using '1-feature' dataset on the test seismic section is about 88.9% while that using '6-feature' dataset is about 89.0%.

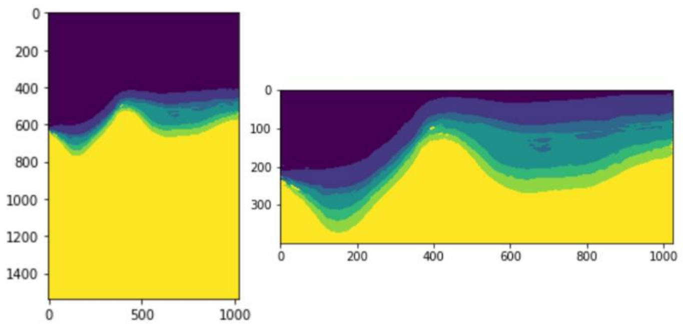


Figure 20. The predicted class label map on test section for encoder-decoder CNN model using '1-feature' dataset and *ResNetUNet 54*.

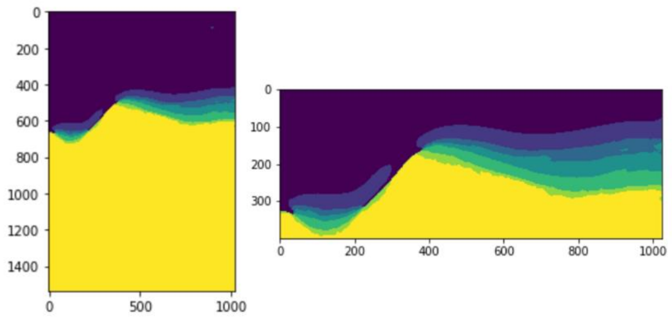


Figure 21. The predicted class label map on test section for encoder-decoder CNN model using '6-feature' dataset and *ResNetUNet 54*.

3.6.5. Conclusion

There is a tradeoff between nice continuity and good accuracy for the encoder-decoder CNN model. Although the class label maps predicted have nice stratigraphic continuity which is good in geosciences, the accuracies of these models on test section are comparatively low.

3.7. Final Ensemble Model

3.7.1. Design

The models trained in those two CNN pipelines need to be stored in a folder at first.

The final ensemble model is implemented with a single function. The first argument of the function is the ‘version’ of this model which is either ‘reflectivity’ for ‘1-feature’ dataset’ and ‘attributes’ for ‘6-feature’ dataset. Then the true class label maps of test section used in both mini-patch CNN model and encoder-decoder CNN model are included. Although the testloaders for ‘1-feature’ and ‘6-feature’ datasets are different. They are fed into the function together for convenience. Next, the models are introduced. After importing indices calibrating the predicted results from encoder-decoder CNN models, this function can work on both ‘1-feature’ and ‘6-feature’ datasets properly.

3.7.2. Results

In this case, I have 3 successful models for both ‘1-feature’ and ‘6-feature’ dataset. The predicted class label maps of the ensemble model with both ‘1-feature’ and ‘6-feature’ datasets on test section are shown in Figure 22 and 23. The final accuracy of the model using ‘1-feature’ dataset on the test seismic section is 96.3%. While that using ‘6-feature’ dataset is 97.3%.

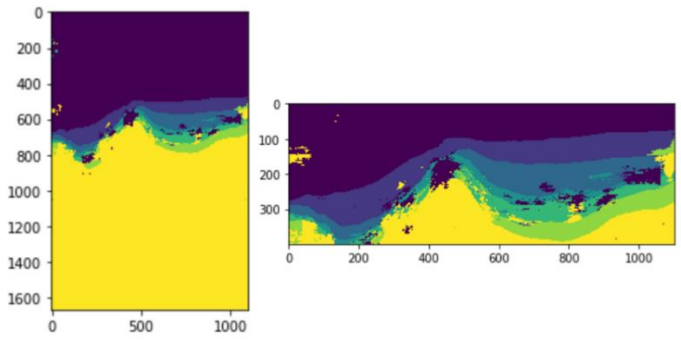


Figure 22. The predicted result on test section for final ensemble model using ‘1-feature’ dataset.

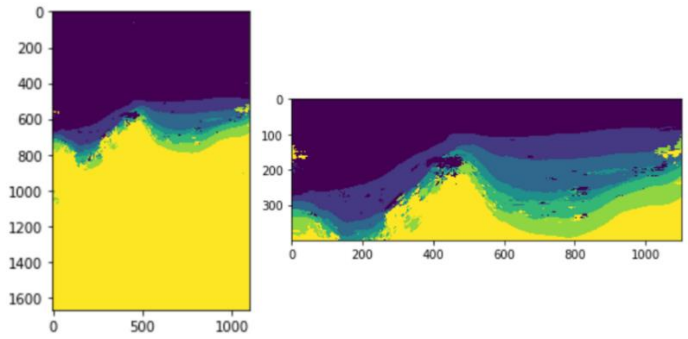


Figure 23. The predicted result on test section for final ensemble model using ‘6-feature’ dataset.

3.7.3. Conclusion

The final ensemble model is designed to absorb the advantages of both the mini-patch CNN model with its good accuracy and the encoder-decoder CNN model with its good ability to indicate continuity. However, pixelization and noise still affect the result. While continuity improve a little, the accuracies of the models trained with both '1-feature' and '6-feature' datasets on test section are worse than those of their best member models.

3.8. Summary

For '1-feature' dataset, the accuracies of the models are shown in Table 4:

	Training	Test
Feed-forward DNN model	N/A	N/A
Mini-patch CNN model (*OptimizedNet*)	99.4%	96.9%
Mini-patch CNN model (*ResNet 34*)	99.7%	95.2%
Encoder-decoder model	N/A	88.9%
Final ensemble model	N/A	96.3%

Table 4. Model accuracies on '1-feature' dataset.

For '6-feature' dataset, the accuracies of the models are shown in Table 5:

	Training	Test
Feed-forward DNN model	35.2%	9.0%
Mini-patch CNN model (*OptimizedNet*)	99.3%	97.7%
Mini-patch CNN model (*ResNet 34*)	99.6%	96.7%
Encoder-decoder model	N/A	89.0%
Final ensemble model	N/A	97.3%

Table 5. Model accuracies on '6-feature' dataset.

4. Discussion

The predicted results of CNN models are good which both reflects in the accuracy score and the predicted class label map for test sections. However, both models have a lot of potential to improve.

I have concluded some reasons for those wrongly classified pixels. The first and the most important reason is artificial noises. The noises can come from each step in seismic data processing and even from data acquisition. Although the models are trained with filtered seismic sections, there may still be some noises (Figure 24). Not only having some artifacts hard to be predicted, the datasets with a lot of noises will ruin the models at the training stage.

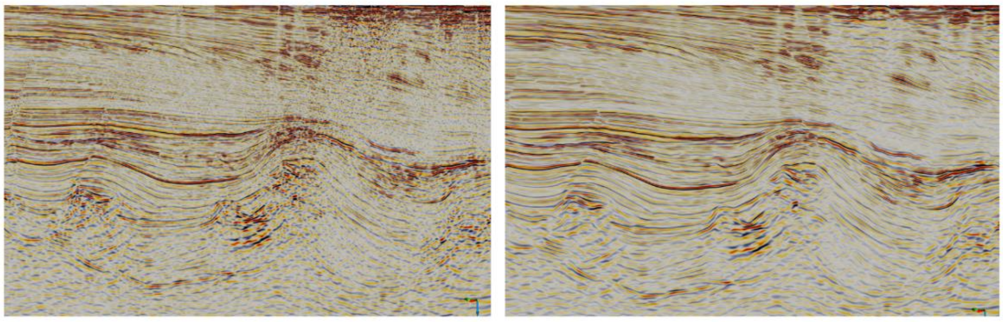


Figure 24. (a) A raw seismic section. (b) The same seismic section after filtered.

The second reason is model design, mainly for mini-patch CNN model. mini-patch CNN model, close to a traditional image classification model, as I mentioned before, has a lot of different network architectures, algorithms and hyperparameters to choose. It is hard to find the best combination of them for a mission in three months.

The two mini-patch CNN models I have implemented, are tested to be the best with both high accuracy and fast convergence speed. However, both models have a lot of potential. From the plots of training vs validation, the accuracy and loss for training datasets of both models keep improving within the whole process. However, those of validation dataset improve fast at the beginning and fluctuate around a certain value in *OptimizedNet* models and the losses of validation dataset even become worse in *ResNet 34* models which indicates an overfitting. Another design of neural network combined with good hyperparameters may help to better against this problem and get better results.

The last reason is the dataset. The distribution of training objects within each class in each seismic section is so imbalanced. For the training dataset, there are originally more than 2.4 million of pixels belong to class 1 and more than 4 million of pixels belong to class 7. However, the most pixels other class has is only about 0.25 million. Even though this problem is solvable in mini-patch CNN model by using data preprocessing algorithm and selection of loss criterion, the model is more or less affected. This problem is serious in encoder-decoder model. The datasets cannot be balanced in the data preprocessing. Even with other attempts to control this problem, the result is still not good enough. This problem is even emphasized with the limited seismic section. I have tried to slice the original seismic sections into small sub-slices horizontally. However, the number of training objects seems still not enough.

The successful models are too few for the final ensemble model. The performance of this model may not be as good as its member models. In this project, for models trained with '6-feature' dataset, the final ensemble model predicts worse than the mini-patch CNN model with *OptimizedNet* by about 0.5% on accuracy for test seismic section. It, as design will always take the result with the most votes, but the votes may not necessarily be as correct as thought. This model will work better with more accurate member models feeding.

The other noticeable point is that the performance of models trained with '6-feature' dataset is proven to be always better than those trained with '1-feature' dataset as expected because the features in the '6-feature' dataset contain frequency-related information which is ignored in the '1-feature' dataset. Although training with '6-feature' dataset takes a little bit longer time, the former models are 0.85% more accurate than the later models on average.

The relation between number of training seismic sections and model performance is positive as well. Training with 1 more seismic section improves the accuracy by 1% on average. Combining with data augmentation, the performance of a model can be greatly elevated.

5. Conclusions

In this project, I successfully build convolutional neural network (CNN) models for supervised seismic facies identification and classification in seismic sections of reflectivity and other featured attributes on the test field in Gabon, Africa. Unseen test seismic sections in the same field can be well predicted. This learning could now be used for either testing human bias on the same field by predicting on all field sections and comparing with human interpreters or running on the other field in the same geological region to see how sensitive the seismic quality variation is the model.

Four different types of model are created in this project which are feed-forward DNN model, mini-patch CNN model, encoder-decoder CNN model and final ensemble model. With feed-forward DNN model as a reference, only successful models from mini-patch CNN model and encoder-decoder CNN model are selected hence create a final ensemble model.

The CNN models work well on the test seismic section. High accuracy score for test dataset doesn't necessarily prove that the model is good. The CNN models trained in this project not only have test accuracies around 90%, but also create the class label maps close to the actual one.

Model trained with more input data are tested to be superior than those with less input data. Using similar training GPU time within same number of epochs, the models trained with more attributes are more accurate than the later models on average. And even with only 1 feature, the model trained with more seismic sections are usually better than that with less seismic sections. Combining these idea and data augmentation, the model can be trained better.

Although the models in this project are trained well, they probably can be optimized with more time.

This project proves that Convolutional Neural Networks (CNNs) can be applied on supervised seismic facies identification and classification. Although only trained with In-line seismic sections on the test field in Gabon, the method can be easily retrained on Cross-lines and other fields as well.

Application of CNNs on seismic section classification is promising and deserve further effort to overcome the proof of concept stage.

Acknowledgement:

This work is the final project report as a part of my completion of Master of Science degree in Applied Computational Science & Engineering at the Imperial College London. I thank Olivier Dubrule, Lukas Mosser, Jeremy Fortun and Kurt Rattansingh for all the supports.

Bibliography:

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., Generative Adversarial Networks, 2014, arXiv:1406.2661.

Goodfellow, I., Y. Bengio, and A. Courville., Deep Learning: MIT Press, 2016.

Kingma, P. D., and Ba, L. J., ADAM: A Method for Stochastic Optimization, 2015, arXiv: 1412.6980v9.

Krischer, L., Megies, T., Barsch, R., Beyreuther, M., Lecocq, T., Caudron, C., and Wassermann, J., ObsPy: a bridge for seismology into the scientific Python ecosystem: Computational Science & Discovery, 8, 014003, 2015, doi:10.1088/1749-4699/8/1/014003.

Krizhevsky, A., Sutskever, I., and Hinton, E. G., ImageNet Classification with Deep Convolutional Neural Network, Advances in Neural Information Processing Systems 25 (NIPS 2012), 2012.

Lin, T., Goyal, P., Girshick, R., He, K., and Dollar P., Focal Loss for Dense Object Detection, 2017, arXiv:1708.02002.

Long, J., Shelhamer, E., Darrell, T., Fully convolutional networks for semantic segmentation, 2014, arXiv:1411.4038.

Milletari, F., Navab, N., Ahmadi, S.A.: V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. In: 2016 Fourth International Conference on 3D Vision (3DV). pp. 565–571. IEEE (oct 2016), 2016.

Ronnerberger, O., Fischer, P., and Box, T., U-Net: Convolutional networks for biomedical image segmentation: Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, 234-241, 2015, https://doi.org/10.1007/978-3-319-24574-4_28.

Rune, H., and Levin, A. S., SEG-Y_r2.0: SEG-Y revision 2.0 Data Exchange format, 2017.

Sudre, H. C., Li, W., Vercauteren, T., Ourselin, S., and Cardoso, M. J., Generalised Dice overlap as a deep learning loss function for highly unbalanced segmentations, 2017, arXiv:1707.03237v3.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A., Going deeper with convolutions: IEEE Conference on Computer Vision and Pattern Recognition, 2015, https://doi.org/10.1109/CVPR.2015.7298594.

Van der Baan, M., and Jutten, C., Neural networks in geophysical applications: Geophysics, 65, no.4, 1032-1047, 2000, https://doi.org/10.1190/1.1444797.

Waldeland, A. U., and Solberg, A. H. S., Salt classification using deep learning: 79th Conference and Exhibition, EAGE, Extended Abstracts, 2017, https://doi.org/10.3997/221446 09.201700918.

Walderland, A. U., Jensen, A. C., Gelius, L., Solberg, A. H. S., Convolutional neural networks for automated seismic interpretation: The Leading Edge 2018 37:7, 529-537, 2018, https://doi.org/10.1190/tle37070529.1.

Wrona, T., Pan, I., Gawthorpe, R. L., and Fossen, H., Seismic facies analysis using machine learning: Geophysics 2018 83:5, O83-O95, 2018, https://doi.org /10.1190 /geo2017-0595.1.

Zhao, T et al., Seismic facies classification using different deep convolutional neural networks, Seismic facies analysis using machine learning: SEG Technical Program Expanded Abstracts 2018. August 2018, 2046-2050, 2018, https://doi.org /10.1190/segam2018-2997085.1.

Zhu, J., Park, T., Isola, P., Efros, A, A., Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, 2017, arXiv:1703.10593.