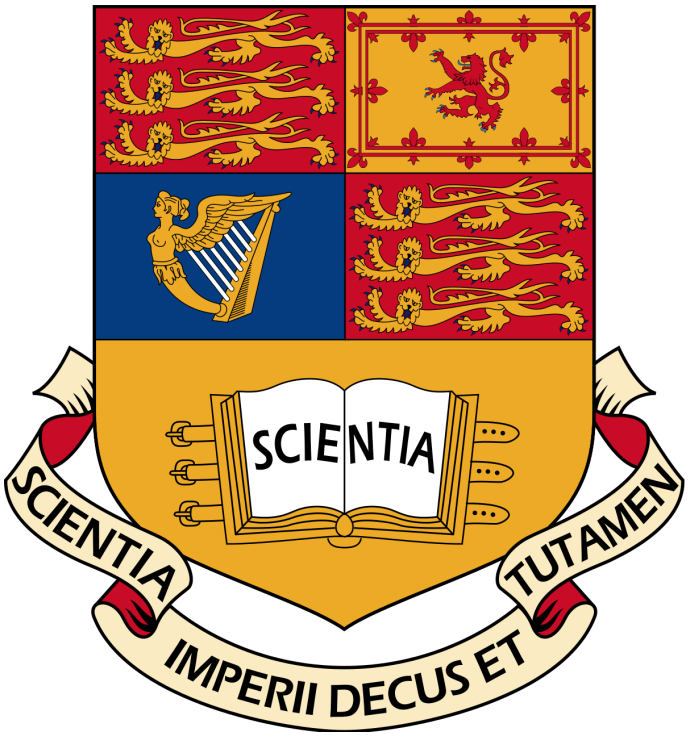
Imperial College London

ACSE9 - Independent Research Project
Applied Computational Science & Engineering
Department of Earth Science & Engineering
Imperial College London

Generative Adversarial Networks applied to multi-component seismic data

Author: Yuxuan Liu
Supervisor: Prof. Michael Warner
Dr. Jiashun Yao
Github: L519159123
Email: yl11718@ic.ac.uk



Abstract

We applied conditional adversarial networks (cGANs) and cycle-consistent adversarial networks (CycleGANs) to the domain translation problem in the field of petroleum geophysics. This simulation makes it possible to generate Z-component geophone data (Z data) from hydrophone data (P data). We demonstrated these two schemes on a small subset of the field data set which was acquired from a four-component ocean-bottom survey over the Tommeliten Alpha field in the North Sea. We explored the optimal configurations of the two networks via the analysis of the hyperparameters and different generator architectures. Three evaluation metrics: mean-squared error, correlation coefficient and structural similarity index have been employed to analyze the test result and to further compare the accuracy of the two networks. Besides, we also explored their feasibility in the real world survey. The experiments proved that both networks were able to generate high quality Z data, with CycleGANs focused more on the accuracy while cGANs require less time and memory. This work suggests we can achieve reasonable Z data by employing only a small number of geophones in the real survey which can greatly save the cost.

Keywords: cGANs, CycleGANs, Z-component geophone data, hydrophone data, domian translation

Code metadata

Current code version	v 1.1
Permanent link to code/repository used for this code version	https://github.com/msc-acse/acse-9-independent-research-project-L519159123
Legal Code License	MIT
Code versioning system used	git
Software code languages, tools, and services used	Python, Pytorch, Scikit-learn, CUDA
Compilation requirements, operating environments & dependencies	Python>=3.6 and Torch>=0.4.1; Torchvision>=0.2.1 and Visdom>=0.1.8.3; Dominate>=2.3.1, Linux and MacOS
If available Link to developer documentation/manual	
Support email for questions	yl11718@ic.ac.uk

1. Introduction

In recent years, many problems in image processing, computer graphics, and computer vision can be posed as "translating" an input image into a corresponding output image. Just as a concept may be expressed in either English or French, a scene maybe rendered as an RGB image, a gradient field, an edge map, a semantic label map, etc. In analogy to automatic language translation, we define automatic image-to-image translation as the task of translating one possible representation of a scene into another, given sufficient training data. Traditionally, each of these tasks has been tackled with separate, special-purpose machinery, despite the fact that the setting is always the same: predict pixels from pixels [8].

The community has already taken significant steps in this direction, with convolutional neural nets (CNNs) becoming the common workhorse behind a wide variety of image prediction problems. However, in recent years, a novel approach - the Generative Adversarial Networks (GANs), gained great popularity in image-to-image translation (Figure 1). GANs learn a loss that tries to classify if the generator output image is real or fake, while simultaneously training the generative network to minimize this loss. Because GANs learn a loss that adapts to the data, they can be applied to a multitude of tasks that traditionally would require very different kinds of loss functions [8].

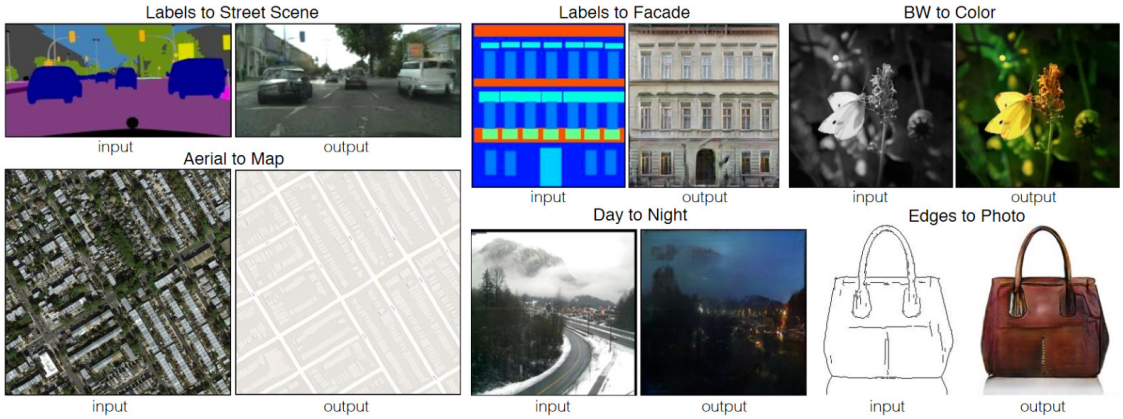


Figure 1: Image-to-Image Translation with Conditional Adversarial Networks [8]

In this project, I mainly explored GANs in the conditional setting. Just as GANs learn a generative model of data, conditional GANs (cGANs) learn a conditional generative model [6]. This makes cGANs suitable for image-to-image translation tasks, where we condition on an input image and generate a corresponding output image. Besides, I also explored the supervised cycle-consistent adversarial networks (CycleGANs), which is regarded as the enhanced conditional GANs as it introduces two more losses - cycle consistency loss and identity loss. Similar approach has been applied to seismic data performing the translations between acoustic domain and elastic domain hydrophone data to enable low-cost elastic data modelling and to remove elastic effects from real observed data before acoustic FWI (Figure 2).

GANs have been vigorously studied in the last few years and many of the techniques I explored in this project have been previously proposed. Nonetheless, earlier papers have barely focused on ocean seismic data applications in petroleum geophysics, and it has remained unclear how effective image-conditional GANs can be as a general-purpose solution for domain translation. Therefore, my primary goal is to demonstrate that on the geophysical domain translation problems, conditional GANs and CycleGANs can produce reasonable results. Furthermore, I also

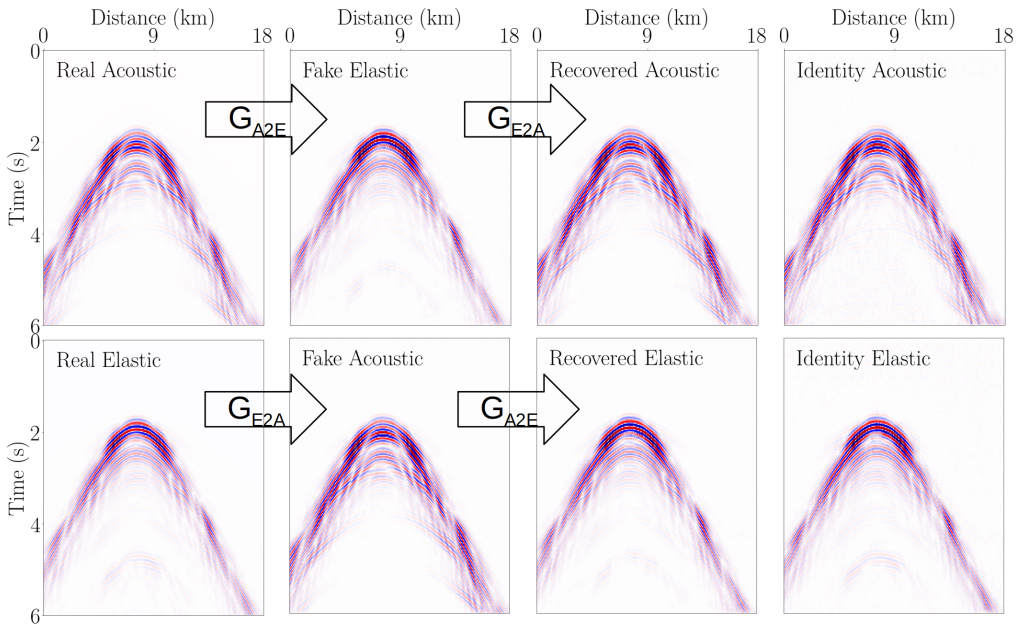


Figure 2: Seismic data translations between acoustic domain and elastic domain with cycle-GAN [27]

explored the optimal architecture and parameter settings with which it can generate accurate and geophysically trustworthy Z-component geophone data (Z data) from hydrophone data (P data).

2. Background

Ocean bottom acquisition is a marine seismic acquisition approaches which is popular in the latest decades [2]. There are two main types of ocean bottom recording: ocean bottom cable (OBC) and ocean bottom node (OBN). In both OBC and OBS acquisition most modern systems make use of four component (4C) sensors, consisting of a 3C geophone and a hydrophone [22]. The hydrophone records only the scalar pressure response and does not distinguish between up-going and down-going wavefields, whereas the geophone records the vector displacement of the seabed that is different for the up-going and down-going wavefield [3]. The source is typically an airgun fired from near the sea surface. This particular seismic acquisition method, known as Ocean Bottom Seismograph (OBS), typically yields superior images of the sub-surface than can be obtained by conventional towed-streamer data, but ocean-bottom data acquisition is significantly more expensive. This project aims to use machine learning method to reduce its acquisition cost.

OBS seismic data provide the information to obtain compressional-wave images of the subsurface. These images usually have fewer water-bottom multiples than streamer seismic data. Because of the nature of the OBS data acquisition, combining the vertical component (Z-component) of the vector wavefield and the pressure component (hydrophone), a process known as PZ summation, produces PZ seismic sections, which are ideally free of water-bottom multiples [1].

In actual experiments, the pressure data is easy to obtain, whereas the acquisition of the vertical velocity wavefields data is relatively expensive and difficult since the geophones are buried into the hole on the seabed individually, usually by a remote operated vehicle (ROV) and with a typical receiver spacing of 100-400m [22], which costs around 20 million dollars for one experiment. Therefore, this project aims to explore the use of machine learning methods in generating the geophone data from

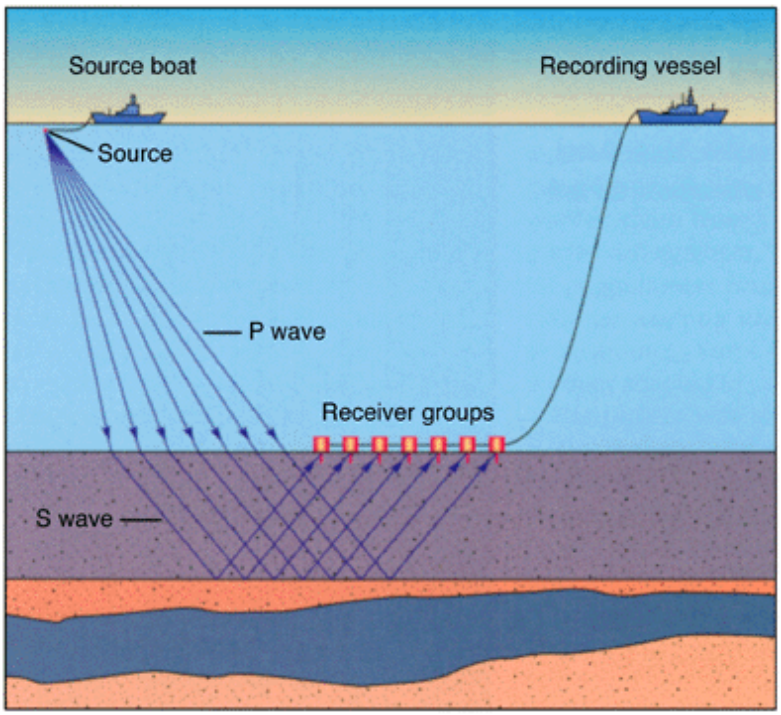


Figure 3: Ocean Bottom Seismograph [4]

hydrophone data without deploying all the geophones. That is to say, translating the seismic data from P (pressure) domain to Z (Z-component velocity) domain via the trained model so as to greatly reduce the data acquisition cost and to make this process more efficient since we only need to collect a small portion of the original data, say 10%, to train the model, and we can generate data with the same features and patterns.

The field data set for this study was seismic data taken from the Tommeliten Alpha field in the North Sea by OBC. Tommeliten Alpha is a gas condensate discovery located 25 km southwest of the Ekofisk field in the Norwegian North Sea, Block 1/9. The reservoir consists of two fractured chalk formations, Ekofisk and Tor, situated at the crest of a broad anticline, approximately 3000 m below the surface. A large part of the reservoir is located in a seismic-obscured area, caused by the presence of gas in the overlying section of interbedded silt and sandstone within the 1000–2000 m depth range [7].

A high-density, full-azimuth, 3D, 4C, ocean-bottom-cable survey was acquired in 2005 with the aim of improving images of the reservoir beneath the gas cloud. The data were acquired using three swaths, each composed of eight parallel cables, in water depths of around 75 m (Figure 4). The cables were 6 km in length; the inline receiver spacing was 25 m, and the crossline spacing between cables was 300 m (Figure 5). Flip-flop shooting using two airgun arrays, each of 3930 cubic inches towed at 6 m depth, was orthogonal to the cables, and used a 75 m cross-track and 25 m along-track separation (Figure 5). For each receiver swath, the shooting patch measured 10 $\times$ 12 km, and together the three patches covered a survey area of about 180 km^2. In total, the survey employed 5760 4C receivers and about 96,000 sources [25].

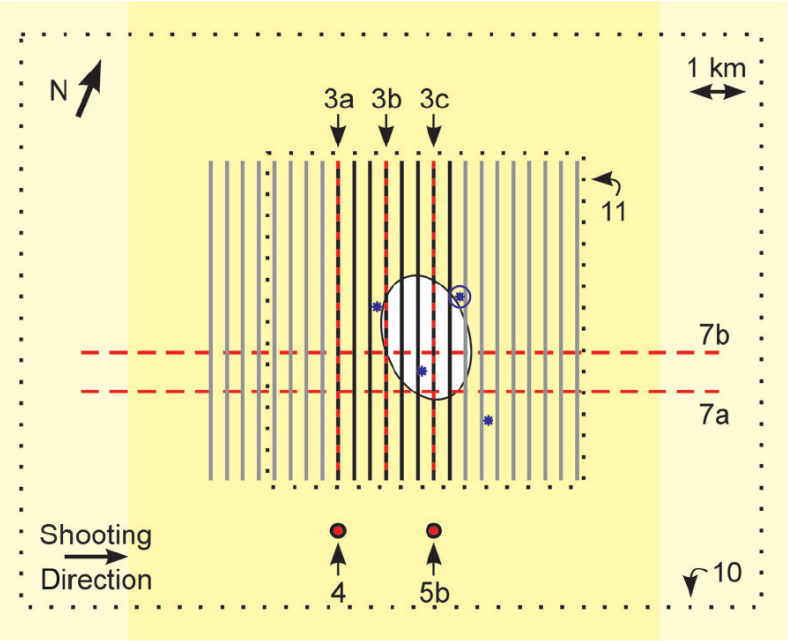


Figure 4: Experimental geometry for the OBC survey across the Tommeliten Alpha field [25].

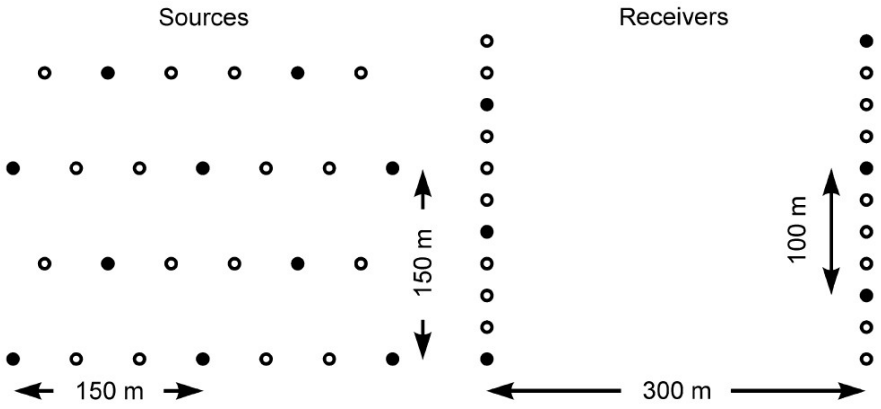


Figure 5: Detail of the source and receiver geometry. Circles show the nominal location of sources and receivers in the original data set [25].

In this report, I first discuss the types of network that I plan to investigate. I follow this with a discussion of the codes I have used for data pre-processing and preparation, for training and applying the networks, and for analyzing and manipulating the results. This is followed by a description of the specific numerical experiments that I have run, and an analysis of the corresponding results. Finally, I end with conclusions and a brief discussion of future work that could continue this project further.

3. Related work

Generative Adversarial Networks (GANs) [6, 28] have achieved impressive results in image generation [5, 16], image editing [29], and representation learning [16, 18, 13]. Recent methods adopt the same idea for conditional image generation applications, such as text2image [17], image inpainting [14], and future prediction [12], as well as to other domains like videos [21] and 3D data [26]. The

key to GANs' success is the idea of an adversarial loss that forces the generated images to be, in principle, indistinguishable from real photos. This loss is particularly powerful for image generation tasks, as this is exactly the objective that much of computer graphics aims to optimize [30].

Image-to-Image Translation A conditional adversarial network architecture with "U-Net"-based generator and convolutional "PatchGAN" discriminator was first proposed by Zhu et al. [8] in 2017, which has been successfully employed for domain transfer problems in computer vision and proved to be effective on a wide range of image domain translation problems. Several months later, their team presented a new network architecture called cycle-consistent adversarial networks [30] to solve domain translation problems in the absence of paired images, which was proved to outperform some traditional adversarial networks architecture. These two networks are regarded as two of the most advanced and reliable methods for domain transfer problems at present and gained great popularity in each field by recent years.

Geophysical Domain Translation with GANs Nam Pham and Sergey Fomel proposed an algorithm for seismic data interpolation using cycle-consistent generative adversarial networks and achieved accurate interpolation results and can easily be applied to 3D seismic data sets [15]. Vincenzo Lipari et al. used a generative adversarial network to process seismic migrated images in order to potentially obtain different kinds of outputs depending on the application target at training stage [11] and showed that it is actually possible to leverage recent findings in deep learning for different geophysical imaging applications. Jiashun Yao et al. used a supervised CycleGAN algorithm to perform forward and reverse mapping between acoustic and elastic data using paired synthetic data [27], which achieved the good results on traditional FWI problem.

4. Method

4.1. Contional GANs

Since in this project, the raw seismic data have already been paired up in P and Z domain manually, this explains why can we apply cGANs to domain translation of the seismic data here.

GANs are generative models that learn a mapping from random noise vector z to output image y, G: $z \rightarrow y$ [6]. In contrast, conditional GANs learn a mapping from observed image x and random noise vector z, to y, G: $\{x, z\} \rightarrow y$. The generator G is trained to produce outputs that cannot be distinguished from "real" images by an adversarially trained discriminator, D, which is trained to do as well as possible at detecting the generator's "fakes". This training procedure is diagrammed in Figure 6 [8]. In this project, the raw P and Z seismic data have already been paired, which enables the direct application of cGANs on them to perform domain translation. Also, the random noise vector z is not employed in pix2pix model.

4.2. CycleGANs

The goal of CycleGANs is to learn mapping functions between two domains X and Y given training samples $\{x_i\}_{i=1}^{N}$ where $x_i \in X$ and $\{y_i\}_{j=1}^{M}$ where $y_j \in Y$. We denote the data distribution as $x \sim p_{data}(x)$ and $y \sim p_{data}(y)$ correspondingly. As illustrated in Figure 7, our model includes two mappings G: $X \rightarrow Y$ and F: $Y \rightarrow X$. However, the mapping F is not important in this project as we only want to generate Z-component geophone data (Z data) from hydrophone data (P data). In addition, we introduce two adversarial discriminators D_X and D_Y, where D_X aims

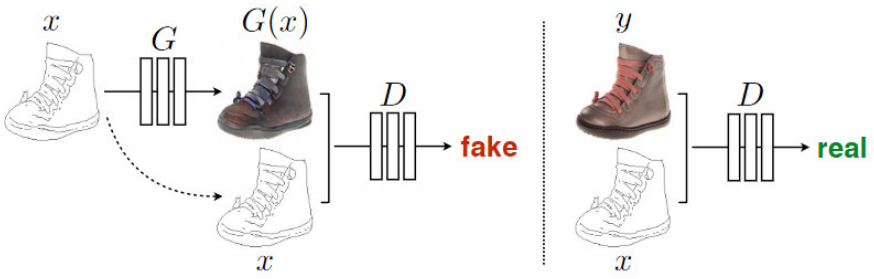


Figure 6: Training a conditional GAN to map edges→photo. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map [8].

to distinguish between images $\{x\}$ and translated images $\{F(y)\}$; in the same way, D_Y aims to discriminate between $\{y\}$ and $\{G(x)\}$. Our objective contains two types of terms: adversarial losses [6] for matching the distribution of generated images to the data distribution in the target domain; and cycle consistency losses to prevent the learned mappings G and F from contradicting each other [30].

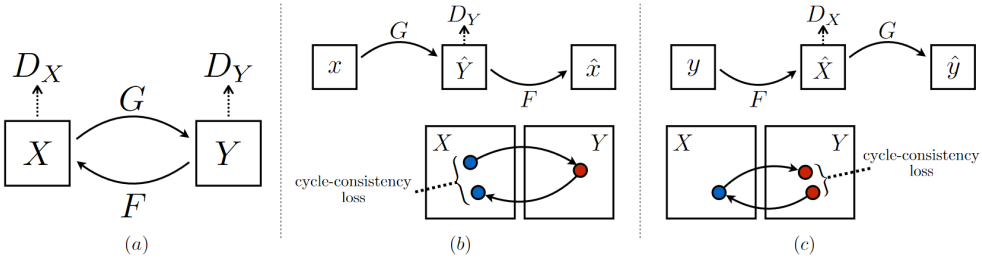


Figure 7: Our model contains two mapping functions G: $X \rightarrow Y$ and F: $Y \rightarrow X$, and associated adversarial discriminators D_X and D_Y [30]. We only consider the mapping G in this project.

5. Software framework

The software is an extension of a pre-existing piece of code (available via github repository `https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix`). The software can train models on paired datasets according to user's options via python script and cache the latest model into disk. Once the model has been trained with `train.py`, we can use the `test.py` script to test the model, then run inference for `--num_test` images and save results to an HTML file and to the corresponding directory. We briefly overview the functionality and implementation of each package and each module on the following sections.

5.1. Data preparing

The raw data can be conventional image files that end with `.JPG`, `.PNG`, `.BMP`, etc. It can also be the text files or binary files. If the former, the data preparation is quite straightforward, we can directly call `combine_A_and_B.py` to pair the data from two domains. If the latter, just like in this project - seismic raw data stored in binary files, we have to call `save_txt` function (Figure 8) via the `data_preprocessing.py` in `data_process` folder to load the binary data and to do the reshaping, transposing and saving process. `save_txt_lines` and `save_txt_crop` are also two important functions used to split and to crop the data according to the experiment purpose. After these steps can we call `combine_A_and_B.py` to pair the data.

```
def save_txt(input_directory, output_directory):
    # iterate over all the data binary file
    for filename in sorted(os.listdir(input_directory)):
        if filename.endswith(".bin"):
            print(os.path.join(input_directory, filename))
            # load data
            with open(os.path.join(input_directory, filename), 'rb') as fh:
                loaded_array = np.frombuffer(fh.read(), dtype="float32")
            # data processing
            img = loaded_array.reshape(240, -1)
            img = img.transpose()

            output = os.path.splitext(filename)[0] + '.txt'
            np.savetxt(os.path.join(output_directory, output), img, delimiter=",")

            fh.close()
            continue
        else:
            continue
```

Figure 8: Example of `save_txt` function

5.2. Dataset structure

`data` directory contains all the modules related to dataset generation. To add a custom dataset class called `dummy`, we need to add a file called `dummy_dataset.py` and define a subclass `DummyDataset` inherited from `BaseDataset`. Four functions needed to be implemented: `_init_` (initialize the class, need to first call `BaseDataset._init_(self, opt)`), `_len_` (return the size of dataset), `_getitem_` (get a data point), and `modify_commandline_options` (add dataset-specific options and set default options). Now we can use the dataset class by specifying flag `--dataset_mode dummy`. Below we explain each file in details.

- `_init_.py` implements the interface between this package and training and test scripts. `train.py` and `test.py` call `from data import create_dataset` and `dataset = create_dataset(opt)` to create a dataset given the option.
- `base_dataset.py` implements an abstract base class (ABC) for datasets. It also includes common transformation functions (e.g., `get_transform`, `_scale_width`), which can be later used in subclasses.
- `image_folder.py` implements an image folder class. We modify the official PyTorch image folder code so that this class can load images from both the current directory and its subdirectories.
- `aligned_dataset.py` includes a dataset class that can load image pairs. It assumes a single image directory `/path/to/data/train`, which contains image pairs in the form of A,B. During test time, we need to prepare a directory `/path/to/data/test` as test data.
- `unaligned_dataset.py` includes a dataset class that can load unaligned datasets. It assumes that two directories to host training images from domain A `/path/to/data/trainA` and from domain B `/path/to/data/trainB` respectively. Then we can train the model with the dataset flag `--dataroot /path/to/data`. Similarly, we need to prepare two directories `/path/to/data/testA` and `/path/to/data/testB` during test time.

- `single_dataset.py` includes a dataset class that can load a set of single images specified by the path `--dataroot /path/to/data`. It can be used for generating CycleGAN results only for one side with the model option `-model test`.

The following snippet (Figure 9) is a example of how we rewrite the `_getitem_` function in the `aligned_dataset.py` so that it can deal with the seismic data text files.

```
def __getitem__(self, index):
    """Return a data point and its metadata information.

    """
    # read a image given a random integer index
    AB_path = self.AB_paths[index]
    AB = np.genfromtxt(AB_path, delimiter=',', dtype='float64').astype('float32')
    # split the paired data
    A = AB[:, :240]
    B = AB[:, 240:]
    # convert array to tensor
    tensor_A = torch.tensor(A, dtype = torch.float)
    tensor_B = torch.tensor(B, dtype = torch.float)
    # add one channel
    A = tensor_A.unsqueeze(0)
    B = tensor_B.unsqueeze(0)

    return {'A': A, 'B': B, 'A_paths': AB_path, 'B_paths': AB_path}
```

Figure 9: Part of `aligned_dataset.py`

5.3. Model architecture

`models` directory contains modules related to objective functions, optimizations, and network architectures. To add a custom model class called `dummy`, we need to add a file called `dummy_model.py` and define a subclass `DummyModel` inherited from `BaseModel`. Four functions need to be implemented: `_init_` (initialize the class; need to first call `BaseModel._init_(self, opt)`), `set_input` (unpack data from dataset and apply pre-processing), `forward` (generate intermediate results), `optimize_parameters` (calculate loss, gradients, and update network weights), and optionally `modify_commandline_options` (add model-specific options and set default options). Now we can use the model class by specifying flag `--model dummy`. Below we explain each file in details.

- `_init_.py` implements the interface between this package and training and test scripts. `train.py` and `test.py` call `from models import create_model` and `model = create_model(opt)` to create a model given the option `opt`. We also need to call `model.setup(opt)` to properly initialize the model.

- `base_model.py` implements an abstract base class (ABC) for models. It also includes commonly used helper functions (e.g., `setup`, `test`, `update_learning_rate`, `save_networks`, `load_networks`), which can be later used in subclasses.

- `pix2pix_model.py` implements the pix2pix model, for learning a mapping from input images to output images given paired data. The model training requires `--dataset_mode aligned` dataset. By default, it uses a `--netG`

`unet256` U-Net generator, a `--netD basic` discriminator (PatchGAN), and a `--gan_mode vanilla` GAN loss (standard cross-entropy objective).

- `cycle_gan_model.py` implements the CycleGAN model, for learning image-to-image translation without paired data, and we can also train it with paired data. The model training requires `--dataset_mode unaligned` dataset. By default, it uses a `--netG resnet_9blocks` ResNet generator, a `--netD basic` discriminator (PatchGAN introduced by pix2pix), and a least-square GANs objective (`--gan_mode lsgan`).

- `networks.py` module implements network architectures (both generators and discriminators), as well as normalization layers, initialization methods, optimization scheduler (i.e., learning rate policy), and GAN objective function (`vanilla`, `lsgan`, `wgangp`).

- `test_model.py` implements a model that can be used to generate CycleGAN results for only one direction. This model will automatically set `--dataset_mode single`, which only loads the images from one set.

Pix2Pix and CycleGAN are two models that we mainly focus on and train with. It is widely known that GANs are hard to train. A major reason for GANs' instability may be that the generative distributions are weird, degenerate, and their support don't generally overlap with the true data distribution. In order to stabilising GAN training, we used a trick here - introducing the Gaussian random noise to both real and fake data during training. Figure 10 snipped from `pix2pix_model.py` shows how we add the random noise to the real and generated images before feeding them into the discriminator. Also, the corresponding experiment result will be presented and analyzed in the next section.

```
def backward_D(self):
    """Calculate GAN loss for the discriminator"""
    # add some random noise between -1 and 1
    val = -1 + 2 * np.random.rand(2, self.real_A.shape[0], 240).astype('float32')
    noise = torch.from_numpy(val).view(1, 2, -1, 240).to(self.device)
    # Fake; stop backprop to the generator by detaching fake_B
    # we use conditional GANs; we need to feed both input and output to the discriminator
    fake_AB = torch.cat((self.real_A, self.fake_B), 1)
    pred_fake = self.netD(fake_AB.detach() + 0.03 * noise)
    self.loss_D_fake = self.criterionGAN(pred_fake, False)
    # Real
    val = -1 + 2 * np.random.rand(2, self.real_A.shape[0], 240).astype('float32')
    noise = torch.from_numpy(val).view(1, 2, -1, 240).to(self.device)
    real_AB = torch.cat((self.real_A, self.real_B), 1)
    pred_real = self.netD(real_AB + 0.03 * noise)
    self.loss_D_real = self.criterionGAN(pred_real, True)
    # combine loss and calculate gradients
    self.loss_D = (self.loss_D_fake + self.loss_D_real) * 0.5
    self.loss_D.backward()
```

Figure 10: Part of `pix2pix_model.py`

5.4. Util structure

`util` directory includes a miscellaneous collection of useful helper functions.

- `_init_.py` is required to make Python treat the directory `util` as containing packages.

- `get_data.py` provides a Python script for downloading CycleGAN and pix2pix datasets. Alternatively, We can also use bash scripts such as `download_data.sh`.

- `html.py` implements a module that saves images into a single HTML file. It consists of functions such as `add_header` (add a text header to the HTML file), `add_images` (add a row of images to the HTML file), `save` (save the HTML to the disk). It is based on Python library `dominate`, a Python library for creating and manipulating HTML documents using a DOM API.

- `image_pool.py` implements an image buffer that stores previously generated images. This buffer enables us to update discriminators using a history of generated images rather than the ones produced by the latest generators. The size of the buffer is controlled by the flag `--pool_size`.

- `visualizer.py` includes several functions that can display/save images, print/save logging information and calculate some quantitative indicators. It uses a Python library `visdom` for display and a Python library `dominate` (wrapped in `HTML`) for creating HTML files with images. As to the quantitative analysis tools, it relies on the Python library `Scikit-learn`.

- `util.py` consists of simple helper functions such as `tensor2im` (convert a tensor array to a numpy image array), `diagnose_network` (calculate and print the mean of average absolute value of gradients), and `mkdirs` (create multiple directories).

The following piece of code (Figure 11) derived from the `visualizer.py` demonstrates how to implement the quantitative analysis in the `save_images` function so that we can calculate the indicators while saving images and do not have to write any other redundant function.

```
if label == 'fake_B':
    with open(save_path2) as f:
        fake_img = np.genfromtxt(f, delimiter=',', dtype='float64').astype('float32')
        f.close()
if label == 'real_B':
    with open(save_path2) as f2:
        real_img = np.genfromtxt(f2, delimiter=',', dtype='float64').astype('float32')
        f2.close()

        # calculate the difference between real and fake imgs
        difference = fake_img - real_img
        # save difference results in txt file
        np.savetxt(save_path3, difference, delimiter=",")
        util.save_image2(difference, save_path4)

        ims.append(img_diff_name)
        txts.append(label2)
        links.append(img_diff_name)
```

Figure 11: Part of `visualizer.py`

5.5. Options structure

Options directory includes our option modules: training options, test options, and basic options (used in both training and test). `TrainOptions` and `TestOptions` are both subclasses of `BaseOptions`. They will reuse the options defined in `BaseOptions`.

- `_init_.py` is required to make Python treat the directory options as containing packages.
- `base_options.py` includes options that are used in both training and test. It also implements a few helper functions such as parsing, printing, and saving the options. It also gathers additional options defined in `modify_commandline_options` functions in both dataset class and model class.
- `train_options.py` includes options that are only used during training time.
- `test_options.py` includes options that are only used during test time.

5.6. Train and test

- `train.py` is a general-purpose training script. It works for various models (with option `--model`: e.g., `pix2pix`, `cyclegan`) and different datasets (with option `--dataset_mode`: e.g., `aligned`, `unaligned`, `single`). It first creates model, dataset, and visualizer given the option. It then does standard network training. During the training, it also visualize/save the images, print/save the loss plot, and save models. The script supports continue/resume training.
- `test.py` is a general-purpose test script. Once we have trained the model with `train.py`, we can use this script to test the model. It will load a saved model from `--checkpoints_dir` and save the results to `--results_dir`. It will hard-code some parameters.

The architectural design of the whole software is diagrammed in Figure 12.

Figure 12: Architectural design diagram

6. Experiments

To explore the GANs' performance on the seismic data domain translation, we test the network on a variety of different dataset configurations:

- P data→Z data, trained and tested on dataset without cropping. This dataset is the original one with 3800 x 240 pixels.
- P data→Z data, trained and tested on cropped dataset. The size of this dataset is 900 x 240 that is cropped from the original dataset.
- P data→Z data, trained and tested on segmented dataset. The size of this dataset is also 3800 x 240 pixels, but its training and test set is split according to cables. The detailed information is provided in *Extension* section.

Details of training on each of these datasets are provided in the supplemental materials online. In all cases, the input and output are simply the 2D array which represent the 1 channel images, so the `--input_nc 1` and `--output_nc 1` are fixed in `train_options` and `test_options`. Qualitative results are shown in Figures 13, 14, 16, 19. More comprehensive results are available via github repository `https://github.com/msc-acse/acse-9-independent-research-project-L519159123/blob/master/report/figure_comparison.pdf`.

Data requirements and speed We note that decent results can be obtained even on small datasets. Our segmented training set consists of 1168 gathers (See results in Figure 21). We trained the network with a single NVIDIA GeForce RTX 2080 Ti GPU and it took around 2 days due to the large image size (3800 x 240). At test time, all models run approximately 20 minutes on 50 test images on this GPU.

6.1. Evaluation metrics

Evaluating the quality of synthesized images is an open and difficult problem [19]. Traditional metrics such as per-pixel mean-squared error do not assess joint statistics of the result, and therefore do not measure the very structure that structured losses aim to capture [8].

To more comprehensively evaluate the visual quality of our results, we employ two more metrics in addition to the fundamental mean-squared error, one is correlation coefficient and another is structural similarity index.

Correlation coefficient (CC) Correlation is a method for establishing the degree of probability that a linear relationship exists between two measured quantities. In 1895, Karl Pearson defined the Pearson product-moment correlation coefficient r. Pearson's correlation coefficient, r, was the first formal correlation measure and is widely used in statistical analysis, pattern recognition and image processing [10]. The correlation coefficient has the value r=1 if the two images are absolutely identical, r=0 if they are completely uncorrelated and r=-1 if they are completely anti-correlated [9].

Structure similarity index metric (SSIM) It is observed that natural image signals are highly structured i.e., the image signal samples exhibit strong dependencies amongst themselves, especially when they are spatially proximate. These dependencies carry important information about the structure of the objects in the visual scene [24]. SSIM is a *full-reference* image quality assessment method; the quality of a test image is evaluated by comparing it with a reference image that is assumed to have perfect quality [23]. The goal of image quality assessment is to design methods that quantify the strength of the perceptual similarity (or difference)

between the test and the reference images. SSIM values can be used to measure the similarity between the structure of a test image and a training image [10].

The combination of mean-squared error, correlation coefficient and structure similarity index provides a more accurate indicator to evaluate the test results.

6.2. Analysis of the hyperparameters

We first trained and tested on dataset without cropping, under default configuration (Table 1). The dataset consists of 1928 training gathers and 408 test gathers, and the size of each gather is 3800 x 240 pixels. Options in `train_options` and `test_options` are: model pix2pix, number of layers of discriminator `(--n_layers_D)` 3, learning rate `(--lr)` 0.0002, the weight for L1 loss `(--lambda_L1)` 100, generator network `(--netG)` resnet_9blocks, discriminator network `(--netD)` 70x70 PatchGAN, number of generator filters `(--ngf)` 64, number of discriminator filters `(--ndf)` 64. The more detailed configurations are listed in Table 1 and the corresponding test result is shown in Figure 13. All the test results presented in this paper are selected from data collected by cable 1 of shot ep0001189.

model	**lambda_L1**	**ndf**	**input_nc**	**netD**
lr	**n_layers_D**	**ngf**	**output_nc**	**netG**
pix2pix	100	64	1	basic
0.0002	3	64	1	resnet9

Table 1: Default configuration

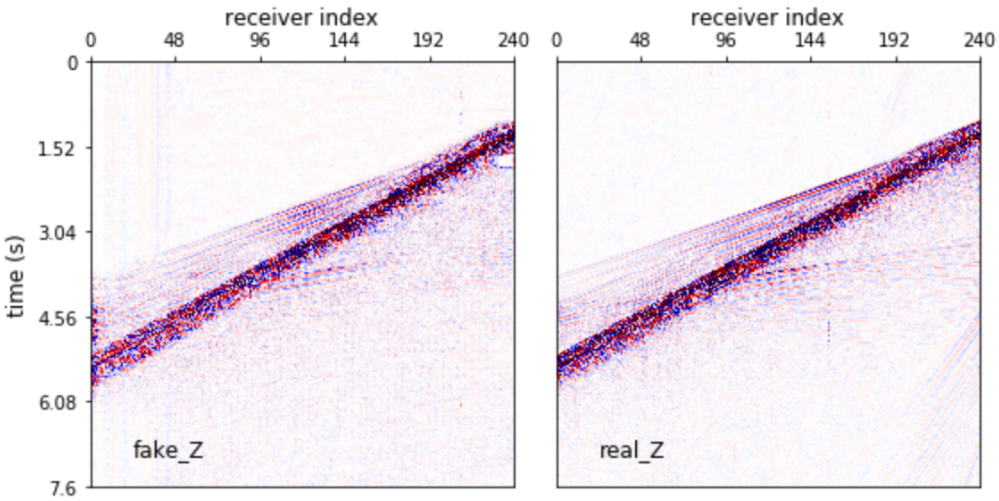


Figure 13: First training experiment under default configuration. The more detailed figure can refer to `https://github.com/msc-acse/acse-9-independent-research-project-L519159123/blob/master/report/figure_comparison.pdf`

The real_Z gather is the real Z data and the fake_Z is generated by cGANs. A significant blank patch appears on the top right corner of fake_Z, a sign that useful energy features from real_Z are being lost in the network. One hypothesis to explain this lapse in accuracy is that the learning rate was set too low and training of the network did not progress to the optimum. Given that 200 epochs of training required 2 days, the learning rate was increased to a value 0.002 while other hyperparameters remain unchanged. Results from the updated network trained using the new learning rate is shown in Figure 14.

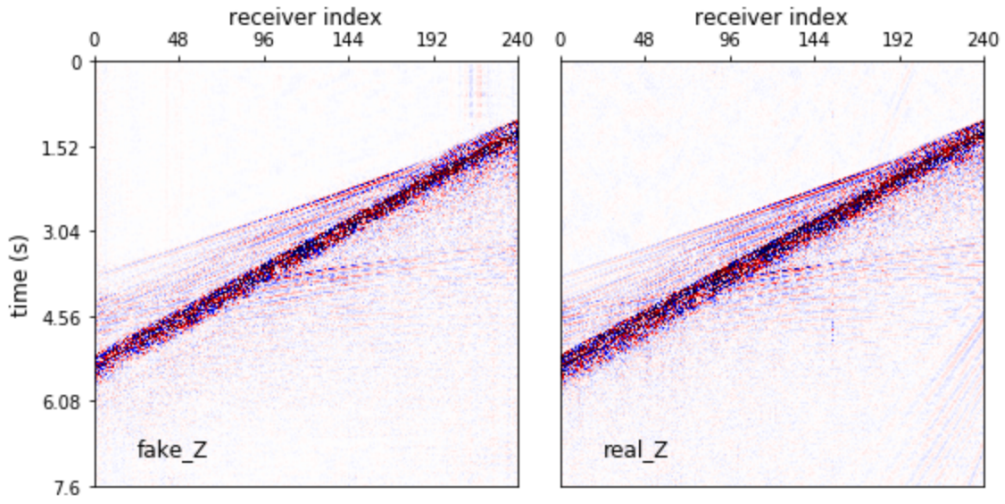


Figure 14: Second training experiment - enlarge learning rate to 0.002

Using the updated learning rate of 0.002 removed the feature loss problem in the upper right corner. Quantitative metrics using the three indicators were calculated from the results before and after the updated learning rate. Table 2 presents all three metrics for averaged based on 50 test images.

	MSE	**CC**	**SSMI**
First experiment	0.00100	0.61514	0.91052
Second experiment	0.00088	0.67858	0.92371

Table 2: Comparison of three indicators scores for first two experiments

From both a visual and quantitative perspective, the updated learning rate improved the accuracy of the network. However, the correlation coefficient is still at a relative low level. The predicted gather still exhibits some blurring and it does not recover some important energy features. For insight into this problem, the loss plot for the entire epoch span is presented in Figure 15.

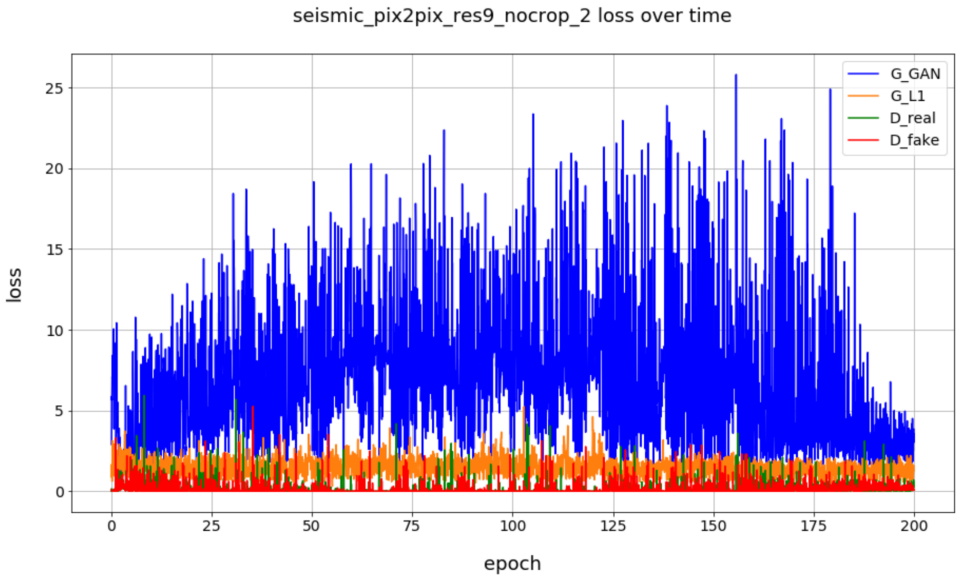


Figure 15: Second training attempt loss over time

The loss plot indicates that discriminator losses reduced to 0 from early epochs,

in contrast with generator losses which fluctuate and only begins to diminish towards the end of training. This signifies that the discriminator can always tell the authenticity of the generated images and prevents the generator from improving. In other words, if the discriminator wins by too big a margin, then the generator can't learn as the discriminator error is too small. To overcome this problem, the discriminator was updated to be less expressive in contrast with the generator, since generation is a more difficult task and requires more parameters. The number of filters in the first convolutional layer of the discriminator was decreased from 64 to 16 and the number of layers reduced from 3 to 1 so as to greatly cripple the power of the discriminator and reduce its number of parameters. Also, to eliminate the blurring influence and to sharpen the edge of energy features, the basic 70x70 PatchGAN discriminator was altered to a pixel discriminator. The configuration for the third experiment is listed in Table 3. Figure 16 and 17 illustrates the corresponding test result and loss.

model	**lambda_L1**	**ndf**	**input_nc**	**netD**
lr	**n_layers_D**	**ngf**	**output_nc**	**netG**
pix2pix	100	16	1	pixel
0.002	1	64	1	resnet9

Table 3: The configuration of hyperparamters in the third experiment

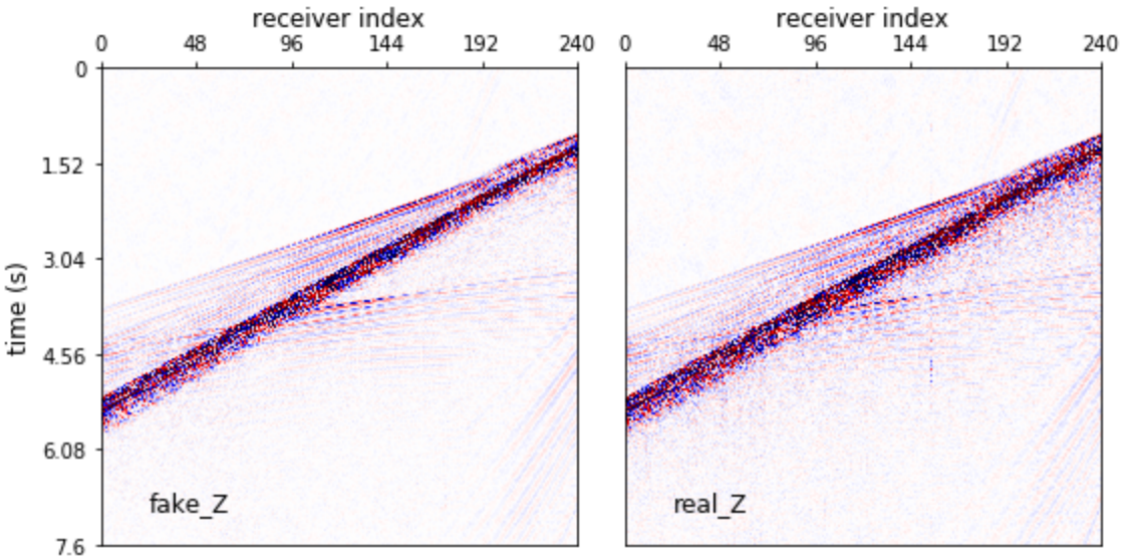


Figure 16: Third training experiment - cripple the discriminator

Visual inspection of the results from the crippled discriminator network indicates improved prediction accuracy. Compared with previous experiments, the predicted image of using a weaker discriminator becomes more clearer and captures most of the features. The trace is smooth and coherent. However, this results was not generated by the latest epoch of the network. Figure 17 shows that although the generator and the discriminator competed with each other quite well for the majority of epochs, they collapsed near the end. The improved predicted gather is taken from the network near the 160th epoch. While switching to a weaker discriminator improved accuracy, it posed a new problem in diverging losses from both the generator and discriminator. This is referred as the GANs' instability, which is caused by

unpredictable generative distributions that generally do not overlap with true data distributions.

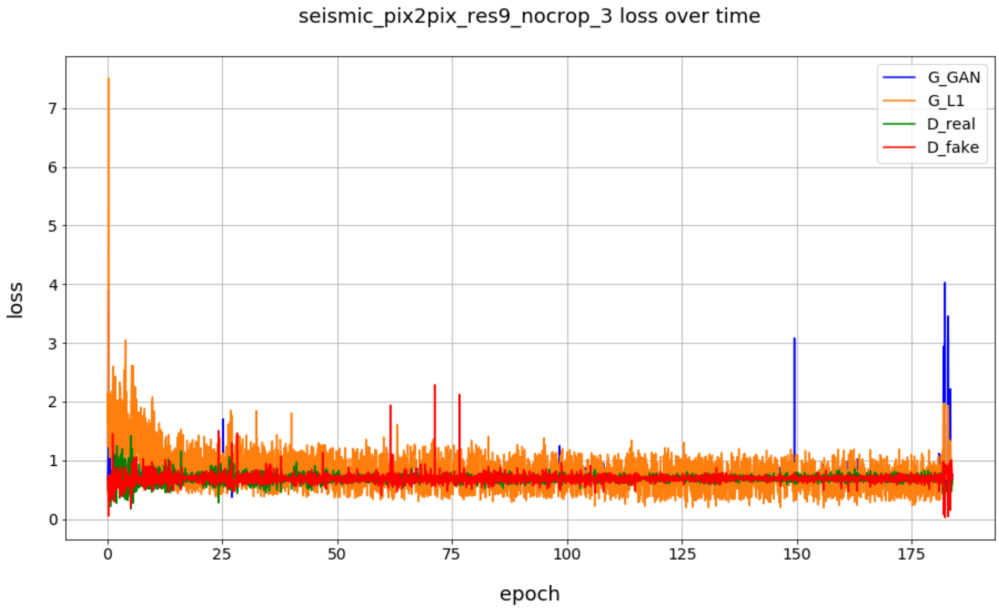


Figure 17: Third training experiment loss over time

Casper Kaae Sønderby et al. suggested adding noise to both real and fake data before feeding them into the discriminator [20] to overcome the problem of instability as this makes the discriminator's job more difficult. From the fourth experiment on, a small Gaussian random noise in the range of -0.03 and 0.03 was introduced into the discriminator (the original pixel value is in the range of -1 and 1). In addition, the weight for L1 loss was increased from the default 100 to 200 since increasing the lambda value strengthens the regularization effect. Jun-Yan Zhu et al. stated using L1 distance rather than L2 encourages less blurring and is effective for super-resolution images [8]. In order to capture the high frequency features of seismic data, the L1 loss term should have a larger weight during training. Table 4 shows the configuration of hyperparameters in the fourth experiment.

model	**lambda_L1**	**ndf**	**input_nc**	**netD**
lr	**n_layers_D**	**ngf**	**output_nc**	**netG**
pix2pix	200	16	1	pixel
0.002	1	64	1	resnet9

Table 4: The configuration of hyperparamters in the fourth experiment

Figure 18 shows that although the loss of the generator and discriminator oscillates significantly during the first 25 epoch, they eventually stabilize during training. The L1 loss dominated the entire training process as expected. The other three losses compete with each other throughout the training process and oscillated in sync, which indicates that the generator and discriminator were in healthy competition.

Table 5 shows that all the three quantitative evaluation metrics improved during the third and fourth experiments, especially the correlation coefficient which increased to 0.87 from an initial value of 0.62. At the same time, the structural similarity index improved to a value 0.97 while the mean square error minimized to 0.00037. Predicted images using the fourth configuration is almost identical to the real test image, both visually and quantitatively.

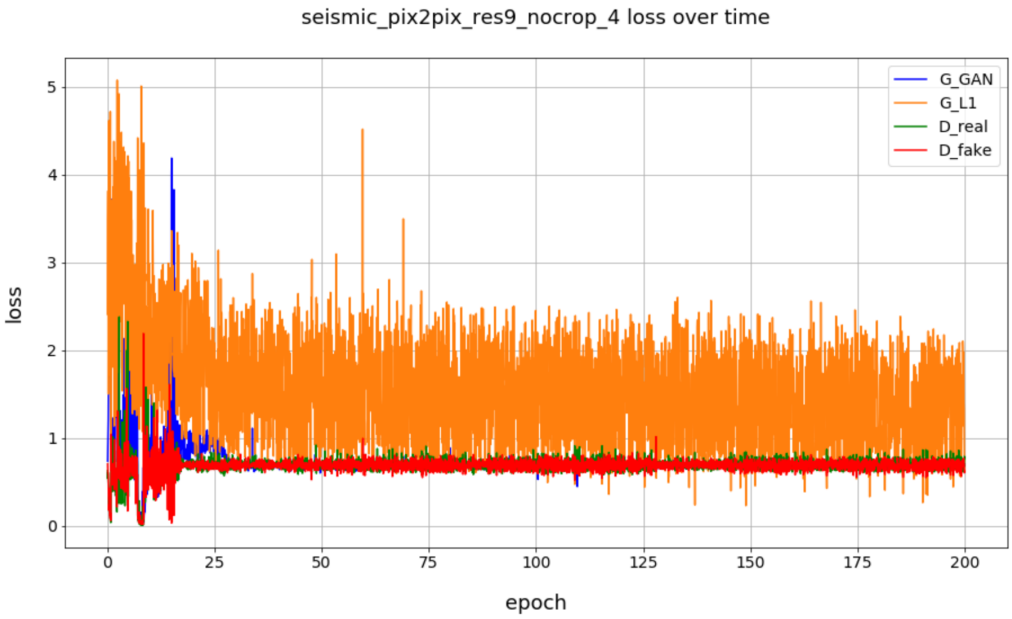


Figure 18: Fourth training experiment loss over time

	MSE	CC	SSMI
First experiment	0.00100	0.61514	0.91052
Second experiment	0.00088	0.67858	0.92371
Third experiment with early stopping	0.00038	0.86809	0.96177
Fourth experiment	0.00037	0.86932	0.96215

Table 5: Comparison of three indicators scores for four experiments

6.3. Analysis of different generator architectures

The previous experiments can be summarized as hyperparameters tuning. Another way to optimize the accuracy of a network is to alter its architecture. The generator architecture we employed in the previous experiments is ResNet with 9 residual blocks (ResNet-9). In the source code, the default generator architecture is a U-Net architecture which allows low-level information to shortcut across the network. However, the limitations of U-Net is that it has a strict requirement for input image sizes - only 256 x 256 or 128 x 128 can be trained using U-Net, which is unsuited to the 3800 x 240 seismic data set required for this project. In order to compare the effects of different generator architecture, we turn to another generator - ResNet with 6 residual blocks (ResNet-6).

Generator	MSE	CC	SSMI
ResNet9	0.00037	0.86932	0.96215
ResNet6	0.00039	0.86236	0.96076

Table 6: Comparison of three indicators scores for ResNet6 and ResNet9

As shown in Table 6, the performance of the three quantitative indicators drop slightly when we switch the generator from ResNet9 to ResNet6. Although ResNet6 still displays high correlation, ResNet9 remains the better choice at this stage.

6.4. Implementation of the CycleGAN architecture

All the above experiments are based on the cGANs. In this subsection, we would explore the performance of CycleGAN, find its optimal configuration, and, compare the result with that of cGANs.

One limitation of adopting the CycleGAN model is that it consists of a circular network that contains doubles the amount of parameters of cGANs. On a 10GB GPU, we face out of memory restrictions when adopting CycleGAN. To overcome this limitation, we cropped the original 3800 x 240 images into smaller image sizes - 900 x 240. The generator architecture remains ResNet9 as it was determined to be the optimal. In the CycleGAN model, we decreased the learning rate and the weight for L1 loss back to the default values of 0.0002 and 100, respectively. We also reduced the number of discriminator filters in the first convolutional layer to 8. The reason why we reduced the number of filters lies in the fact that the cycle process in the CycleGAN enhances the performance of the discriminator, which reverts back to our initial problem. Additionally, CycleGAN introduces two new loss metrics, identical loss and cycle consistency loss, which partly mimic functionality of the original L1 loss. The remaining options continue to follow the optimal configuration of cGANs. Detailed configuration of hyperparameters is shown in Table 7.

model **lr**	**lambda_L1** **n_layers_D**	**ndf** **ngf**	**input_nc** **output_nc**	**netD** **netG**
cycle_gan	100	8	1	pixel
0.0002	1	64	1	resnet9

Table 7: The configuration of hyperparamters in the CycleGAN experiment

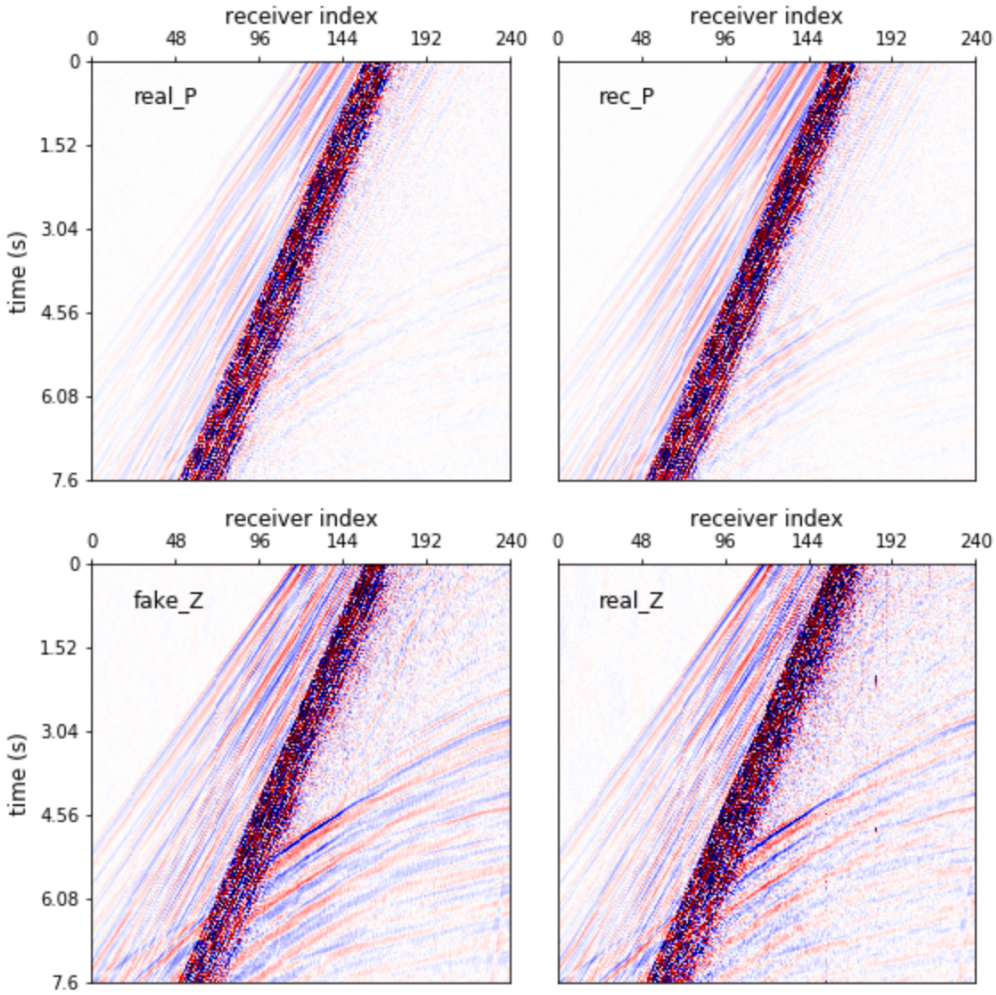


Figure 19: Fifth training experiment with CycleGAN: real_P→fake_Z→rec_P. We only focus on the accuracy of fake_Z gather in this project.

Figure 19 shows that after cropping, the network focused more on the key features of images. The fake_Z gather predicted the features in real_Z gather, and the rec_P gather which is reverse generated by fake_Z gather also recovered the features of real_P with high accuracy. The losses shown in Figure 20 also reflects this improved accuracy. The discriminator loss and the generator loss compete and oscillate around 0.4 while the identical loss and the cycle consistency loss are reduced to the significantly low levels. All the losses decrease over time and the amplitude of the losses also reduce during training process. It appears that the two extra losses introduced by CycleGAN sets a more strict constraint on the convergence of the training process and encourages better results.

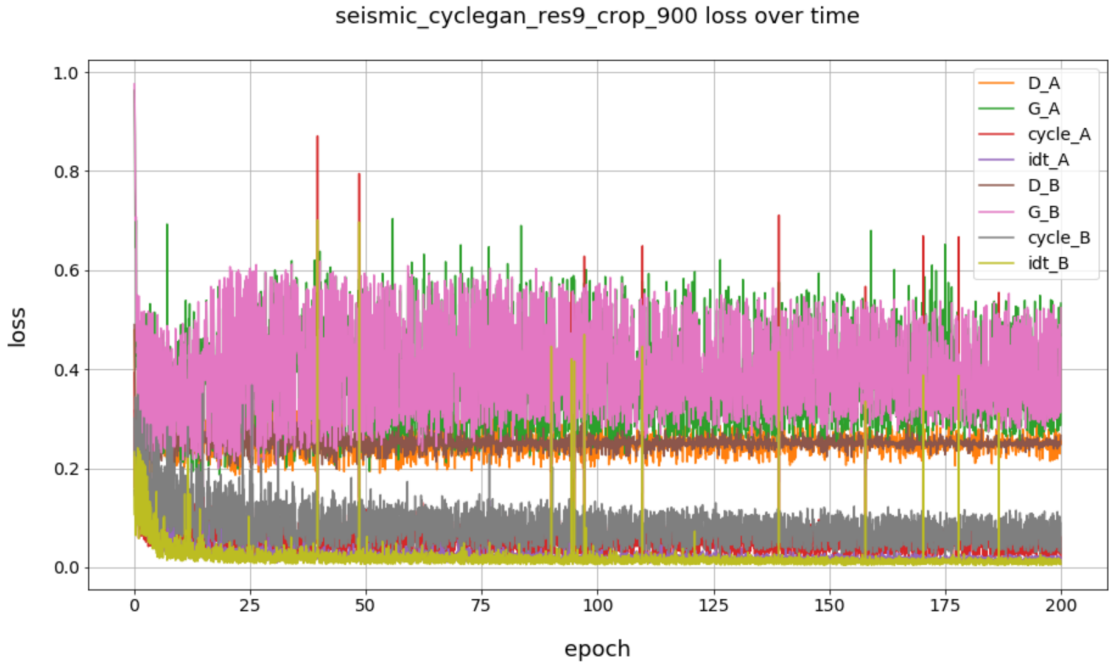


Figure 20: Fifth training experiment loss over time

For verification, we used the cropped image data set on the cGANs model. The updated results of the cGANs model on the cropped dataset is presented in Table 8. The results indicate that under optimal configurations, the CycleGAN model behaves better than the cGAN model for each quantitative metric. However, this minor improvement is established on a significant sacrifice in memory usage and time consumption. Therefore, balancing the trade-off between prediction accuracy and prediction costs requires will depend on the requirements of the problem.

Generator	**MSE**	**CC**	**SSMI**
CycleGAN	0.00069	0.87654	0.92962
cGAN	0.00074	0.86216	0.92619

Table 8: Comparison of three indicators scores for CycleGAN and cGAN

6.5. Extension

As we mentioned in the first section, the data were acquired using three swaths, each composed of eight parallel cables, shown in Figure 21.

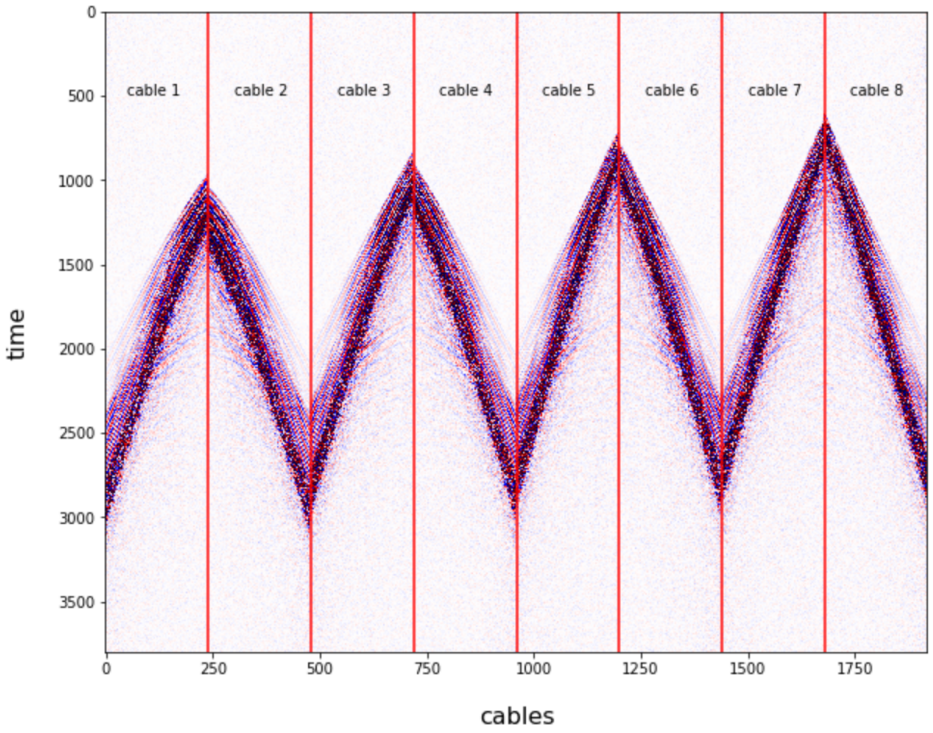


Figure 21: Experimental geometry of eight parallel cables

For all the previous experiments, the training set and test set images contained data from all the sections divided by these eight cables. However, in order to save cost in real world surveys, not all of the cables will be deployed. Ideally, only half of the cables will be deployed, for instance, cable 1 & 3 & 5 & 7, and data from this half will be used to train and predict the missing data of cables 2 & 4 & 6 & 8. The following experiment is based on this idea of using only half the training and test set. When testing for the missing cables, the data is flipped. In order to keep the structural integrity of the original image, we select the cGANs to undertake this experiment. The hyperparamters selected follow the corresponding optimal configurations we explored in the last section that is shown in Table 4.

	MSE	**CC**	**SSMI**
cGAN split in cables	0.00048	0.85472	0.95144
cGAN	0.00037	0.86932	0.96215

Table 9: Comparison of three indicators scores for cGAN with and without splitting cables

Results in Table 9 indicate the performance of the cGANs model trained on the split dataset dropped, which may be explained by amplitude differences between the symmetrical neighboring cables. Generally, the accuracy of the predicted image is still impressive, with a 0.85 correlation coefficient and 0.95 structural similarity, which further proves the feasibility of the commercial applications of this model to the geophysics community.

7. Conclusion

The results in this paper suggest that both conditional adversarial networks and cycle-consistent adversarial networks are promising approaches for many domain translation tasks in petroleum geophysics. These networks learn a loss adapted to the task and data at hand, which makes them applicable in a wide variety of settings. The cycle-consistent adversarial networks achieve comparatively higher accuracy while the conditional adversarial networks require less training time and memory. Selecting the appropriate network therefore requires consideration as it involves a trade-off between the prediction accuracy and prediction cost. The final extended experiment and the corresponding results prove that both methods are well suited for industrial applications. The costs associated with traditional ocean-bottom cable surveys can be largely mitigated using these networks, which serves a great commercial benefit.

8. Future work

The most difficult tasks to resolve in completing the implementation is tuning hyperparameters. As is known to known, the generative adversarial networks is very difficult to train, and we have to keep trying different configurations to ensure its stability. Therefore, one of the future work is how to determine the appropriate beginning parameters. Another future work is to optimize the memory usage and time consumption so as to accelerate the whole training process. Last but not least, although we use the perceptual validation and introduce three quantitative analysis indicators to evaluate the result of the predicted image, it still can not prove that we capture the majority of the useful features. The most authoritative and convinced method is to use the traditional geophysical interpretation to analyze the result. However, it need the professional knowledge and skills of the geophysicist and can not be accomplished in the short time.

Acknowledgments: I would first like to thank my project supervisor Professor Mike Warner and Doctor Jiashun Yao of the Department of Earth Science and Engineering at Imperial College. Professor Mike Warner provided me with the brand new powerful device to do the machine learning research, and he always patiently taught me the geophysical knowledge on the meeting of every week. The door to Doctor Jiashun Yao's office was also open whenever I ran into a trouble or had a question about my research or writing. They consistently believed in my ability to make some achievements, and steered me in the right direction whenever they thought I needed it.

I would also like to express my very profound gratitude to my parents and to my girlfriend for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this paper. This accomplishment would not have been possible without them. Thank you all and I will be grateful forever for your love.

References

[1] http://sep.stanford.edu/sep/daniel/daniel_thesis/paper_html/node1.html.

[2] https://wiki.seg.org/wiki/Ocean-bottom_node.

[3] https://www.slb.com/services/seismic/geophysical-processing/multicomponent/pz-processing.aspx.

[4] http://www.peakseismic.com/content/ocean-bottom-seismic.asp.

[5] E. L. Denton, S. Chintala, R. Fergus, and et al. Deep gen-erative image models using a laplacian pyramid of ad-versarial networks. *NIPS*, 2015.

[6] I. Goodfellow, J Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *NIPS*, April 2014.

[7] Granli, J. R., B. Arntsen, A. Sollid, and E. Hilde. Imaging through gas-filled sediments using marine shear-wave data. *Geophysics*, June 1999.

[8] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-Image translation with conditional adversarial networks. *Computer Vision and Pattern Recognition*, November 2016.

[9] Eugene K. Jen and Roger G.Johnston. The ineffectiveness of correlation coefficient for image comparisons.

[10] Avneet Kaur, Lakhwinder Kaur, and Savita Gupta. Image recognition using coefficient of correlation and structural similarity index in uncontrolled environment. *International Journal of Computer Applications*, 2012.

[11] Vincenzo Lipari, Francesco Picetti, Paolo Bestagini, and Stefano Tubaro. A generative adversarial network For seismic imaging applications. *SEG*, 2018.

[12] M. Mathieu, C. Couprie, and Y. LeCun. Deep multiscale video prediction beyond mean square error. *ICLR*, 2016.

[13] M. F. Mathieu, J. Zhao, A. Ramesh, P. Sprechmann, and Y. LeCun. Disentangling factors of variation in deep representation using adversarial training. *NIPS*, 2016.

[14] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. *CVPR*, 2016.

[15] Nam Pham and Sergey Fomel. Seismic data interpolation using CycleGAN. *SEG*, 2019.

[16] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *ICLR*, 2016.

[17] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. *ICML*, 2016.

[18] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training GANs. *NIPS*, 2016.

[19] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. *NIPS*, 2016.

[20] Casper Kaae Sønderby, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszár. Amortised map inference for image super-resolution. *ICLR*, 2016.

[21] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating videos with scene dynamics. *NIPS*, 2016.

[22] Yi Wang, Sergio Grion, and Richard Bale. What comes up must have gone down the principle and application of up-down deconvolution for multiple attenuation of ocean bottom data. *RECORDER*, December 2009.

[23] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. Image quality assessment: from error visibility to structural similarity. *Proceedings of IEEE Transactions on Image Processing*, 2004.

[24] Zhou Wang, Alan C. Bovik, and Hamid R. Sheikh. Structural similarity based image quality assessment. *Digital Video Image Quality and Perceptual Coding*, 2005.

[25] Michael Warner, Andrew Ratcliffe, Tenice Nangoo, and et al. Anisotropic 3D full-waveform inversion. *Geophysics*, March 2013.

[26] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *NIPS*, 2016.

[27] J. Yao, L. Guasch, M. Warner, D. Davies, and A. Wild. Removing elastic effects in fwi using supervised cycled generative adversarial networks. *EAGE*, May 2019.

[28] J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. *ICLR*, 2017.

[29] J.-Y. Zhu, P. Krahenbuhl, E. Shechtman, and A. A. Efros. Generative visual manipulation on the naturalimage manifold. *ECCV*, 2016.

[30] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *ICCV*, March 2017.