Automatic Seismic Interpretation Techniques

Tayfun Karaderi (CID: 01062606)

Department of Earth Sciences, Imperial College London, United Kingdom

Supervisors: Ehsan Naeini (Ikon Science), Olivier Dubrule & Lukas Mosser (Imperial)

Traditional seismic interpretation techniques in commercial software often rely on choosing a set of attributes and then training a classifier algorithm to discriminate between different classes. We investigate more elegant architectures of deep learning, which have the advantage of combining attribute extraction and classification in a single network. These techniques have also been shown to produce state-of-the-art results in semantic segmentation tasks. We review the performance in terms of prediction accuracy and computational cost of fully convolutional networks (FCNs) and Sliding Window Classifiers (SWCs) on seismic datasets. We also investigate conditional random fields (CRFs) as post-processing tools to structure the output space of deep learning techniques.

Key Words: Seismic Interpretation, Semantic Segmentation, Deep Learning, Fully Convolutional Networks (FCNs), Sliding Window Classifiers (SWCs), Conditional Random Fields (CRFs)

I. Introduction

Automatic interpretation has long been an active topic of research and innovation for seismic exploration and reservoir characterisation workflows. Over the years various algorithms have been introduced and implemented for interpreting salt bodies [1], first breaks [2], horizons [3], channel bodies and faults [4]. However, most of the practical implementations, especially in commercial software, often rely on choosing a set of attributes in advance and then training the classification algorithm to discriminate between different classes. This process still involves human tuning and validations which ultimately means such methods are time-consuming.

In this project, we propose to use supervised deep learning methods for automatic interpretation which has the advantage of combining attribute extraction and classification in one network. Our main objective is to analyse performance in terms of prediction accuracy and computation time of various deep learning techniques (e.g. sliding window classifier and fully convolutional networks) as well as various neural network architectures (e.g. Auto-encoders and U-Net) for seismic image segmentation. Then, the best technique for seismic image segmentation can be incorporated into a geoscience workflow so that a generalist geoscientist can make use of the machine

learning method to train a model from a small subset of labelled seismic data and predict the labels for the entire volume of their seismic dataset with ease, not worrying about writing custom software for each use case.

In this project, we make use of the seismic dataset containing 401 inline slices of size 401x251 pixels obtained from the Forties oil field in the North Sea (UK production block 21/10) by BP. We also tested our implemented models on a salt dataset containing 760 inline slices of size 1461 to 761 collected from the Gulf of Mexico (see appendix I). Google-Colab was used as the development ecosystem throughout the project.

I.I Background Theory

Conventional feed-forward neural networks (NNs) are commonly used in classification problems where the user has to provide a set of attributes and the network will assign a class. NNs consist of many layers of neurons, where the first layer is fed with the user extracted attribute vector and the successive layers are fed with the output from the previous layers. Each neuron (i), in a given layer (j), operates on the input vector (**x**) by weighing (w_i) and summing each element plus adding a bias term (b_i):

$$y_i = \sigma\left(\sum_j w_{i,j}x_j + b_i\right), \qquad (1)$$

where σ is the non-linear activation function. The values of the weights and the bias terms are learned by the traditional back-propagation algorithm [5] that minimizes the loss function which is often a metric of distance between the predicted and the actual labels.

In convolutional neural networks (CNNs), the input attribute vector is replaced by the image data and the weight and sum operations are replaced by the filtering operations. Each neuron (i), now performs a convolution with the kernel ($\mathbf{w_i}$) and the input data (**x**):

$$y_i = \sigma(\boldsymbol{w_i} * \boldsymbol{x_i} + b_i), \qquad (2)$$

The output of each layer is downsampled by stride to reduce the spatial dimensionality. The network often has an increasing number of output channels going down the down-sampling path to extract more attributes. After extracting the desirable attributes, the output of the last *convolutional layer* is vectorized and fed to some conventional NN layers called *fully connected layers* at the end of the network. Traditionally, the last *fully connected layer* of the network has the same number of nodes as the number of classes and a SoftMax activation function is applied to the final layer to form a distribution across different classes and the class with the highest probability is chosen as the prediction. The CNNs are also trained by using the back-propagation algorithm to learn the filter weights.

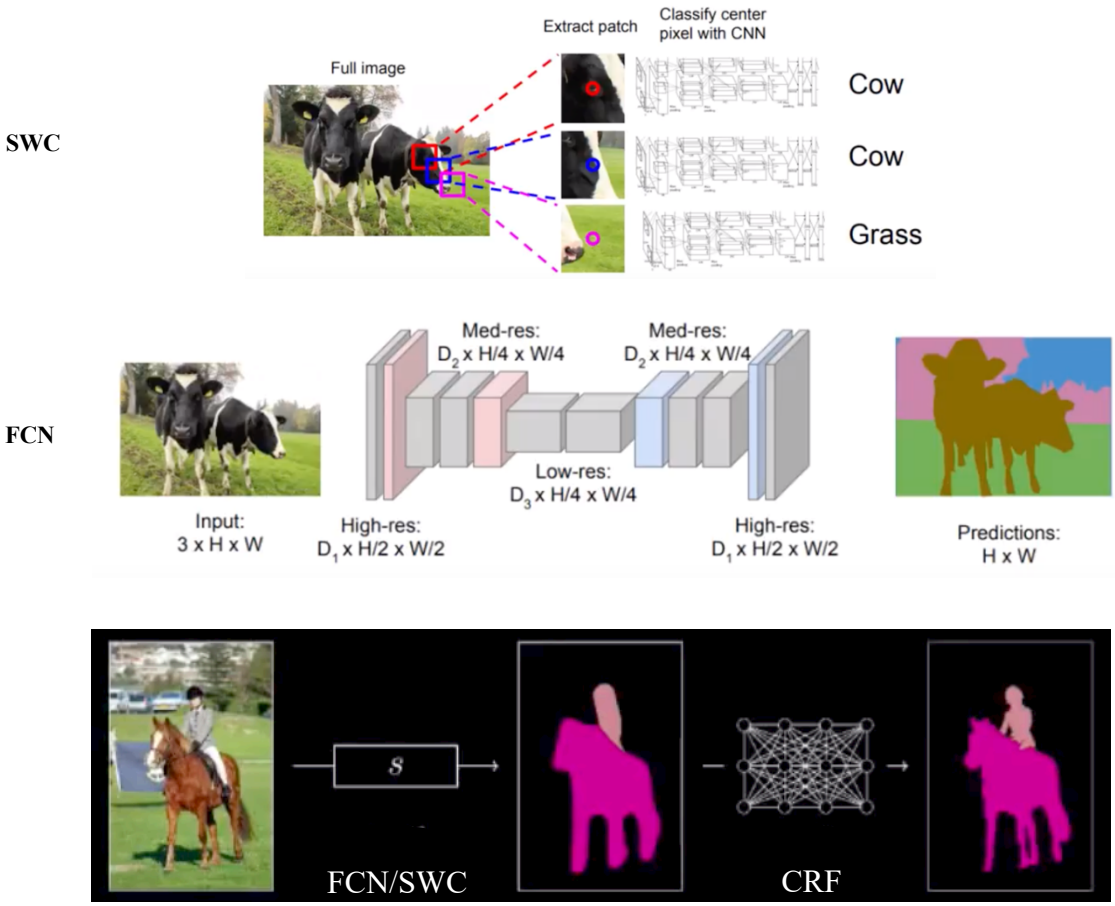

Figure 1: A comparison of the FCN and the SWC approaches. SWCs extract patches for each pixel and labels each patch using a CNN. FCNs on the other hand, map the input image directly to a segmented image of the same size in one go. The last figure shows a CRF model applied to the segmented image to structure the output space.

For a semantic segmentation task where the aim is to predict for each pixel the class it belongs to, the most traditional methods typically relies on either a sliding window classifier (SWC) or a fully convolutional network (FCN) as displayed in figure 1.

SWCs were first used by Ciresan et al. [6] and won the EM segmentation challenge at ISBI 2012. This approach extracts patches from the image and then uses a CNN to label the centre pixel over and over again to classify each pixel in the dataset. This results in training data in terms of patches which is much larger than the number of training images. This is of significant convenience in fields such as seismic interpretation where the number of training images is often very limited since manually labelling seismic datasets tend to be a time-consuming and expensive process. However, despite its advantages on training, this method results in high computational cost for making predictions where often many thousands or millions of patches would have to be extracted and labelled to obtain the segmentation map of a single image. Another drawback of this method is that it often requires padding to label the pixels near the edges of the image when the distance

between the pixel and image edge is less than that of the half patch size. This often imply that the predicted labels on the edges of the image may be less accurate than the other parts.

Fully Convolutional Networks (FCNs) on the other hand, are the more elegant architectures as they avoid redundant computation of low-level filters many times on pixels in overlapping patches. A typical FCN architecture involves a down-sampling path achieved through pooling operations, which increases the receptive field size in order to capture more context. This down-sampling path is often accompanied by an increasing number of output channels to extract more attributes from the input images. However, despite gaining context, the resolution of the output maps is reduced and hence localization of the context is lost. For this reason, the down-sampling path is often followed by a symmetric up-sampling path achieved by transposed convolution operations which learn the localization of the context in low-resolution kernel maps. Sideway connections which preserve spatial information are also present in the most state-of-the-art FCN architectures such as the U-Net architecture (figure 2) of Ronneberger [7] which won the ISBI cell tracking challenge in 2015. Since the U-Net architecture has been proven to be so successful in semantic segmentation tasks we chose to use a U-Net architecture very similar to that of Ronneberger as our FCN model (see appendix II for the details of the U-Net architecture used).

Although deep neural networks (DNNs) have been proven to excel at many computer-vision tasks, many state-of-the-art segmentation algorithms also include a CRF model within their pipelines. One common feature of DNNs is that despite their ability to learn rich feature representations, they predict label variables (pixels) independently from each other. CRF models, on the other hand, belong to the family of probabilistic graphical models which are very powerful tools for modelling the correlations between the pixels being predicted. For this reason, combining the CRF models with DNNs can result in better predictions by acknowledging that we are predicting a structured output and explicitly modelling the problem to include our prior knowledge about the spatial image architecture. In our pipeline, we consider fully connected CRFs with Gaussian edge potential [8] as a post-processing tool for our DNN prediction as displayed in figure 1. We make use of the highly efficient approximate inference algorithm of Krahenbuhl et al. [8] which is publicly available under MIT Licence on their Github repository [9]. The CRF model considers the Gibbs energy,

$$E(\boldsymbol{x}) = \sum_i \psi_u(x_i) + \sum_{i<j} \psi_p(x_i, x_j)\,, \qquad (3)$$

where ψ_u is the unary-potential term which describes the cost of assigning a certain label to a given pixel. This term is taken as the reciprocal of the class proportion. This is obtained from our CNN/FCN predicted segmentation maps which

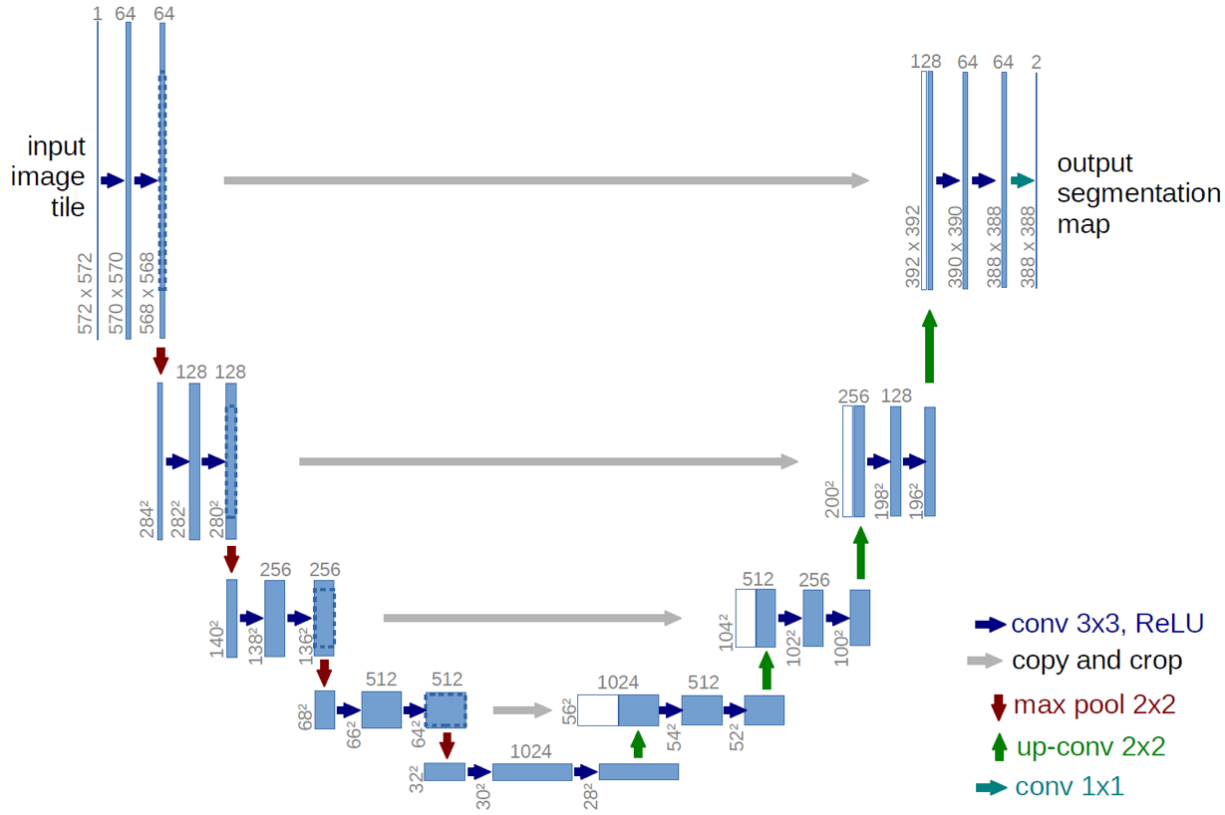

Figure 2: The U-Net model that won the 2015 ISBI cell tracking challenge. Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y scale is provided at the bottom left edge of the box. White boxes represent copied feature maps from the sideway connections.

we feed into our CRF model. The term ψ_p is the pairwise gaussian potential term which forces nearby pixels to be in the same class to make the objects in our segmentation map more continuous. The pairwise potential has the form,

$$\psi_p(x_i, x_j) = \mu(x_i, x_j)k(\boldsymbol{f}_i, \boldsymbol{f}_j), \qquad (4)$$

where, $\mu(x_i, x_j)$ is the compatibility function given as,

$$\mu(x_i, x_j) = \begin{cases} 0 \ [x_i = x_j] \\ 1 \ [x_i \neq x_j] \end{cases}. \qquad (5)$$

$\mathbf{f}_i$ and $\mathbf{f}_j$ are position vectors for pixels i and j, and k is a gaussian kernel given by,

$$k(\boldsymbol{f}_i, \boldsymbol{f}_j) = w \exp\left(-\frac{|\boldsymbol{f}_i - \boldsymbol{f}_j|^2}{2\theta^2}\right), \qquad (6)$$

where w and θ are parameters which are set by the user. Then, the job of the inference algorithm as implemented by Krahenbuhl [9] is to find the configuration of the segmentation map that minimizes the Gibbs energy in equation 3. This way we penalize for the nearby pixels having different labels and thus make the object boundaries in our predictions smoother and more continuous as shown in figure 3.

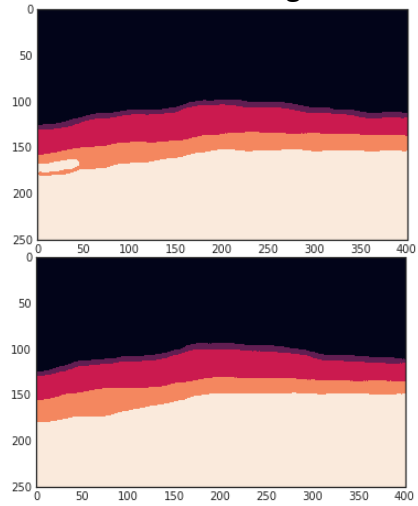

Figure 3: The top figure shows a segmentation map of a seismic inline obtained from an FCN model. The bottom figure shows the segmentation map after the CRF post-processing.

I.II A Brief Literature Review

In the last few decades, it was shown that simple neural network architectures such as MLPs (multi-layer perceptrons) can be used for automatic seismic interpretation techniques such as to segment seismic facies in 3D volumes [10, 11, 12]. However, these methods heavily relied on choosing multiple attributes in advance and, only very recently, the next generation of machine learning techniques such as CNNs (convolutional neural networks) have been successfully applied for automatic seismic interpretation (e.g. by Waldeland et al. [1, 13] and by Badrinarayan et al [14] (Figure 4). This transformation of automatic interpretation techniques has been empowered by algorithmic advances in the field of machine learning (e.g. developments of CNNs by Krinzhevsky in 2012 [15], GANs by Goodfellow in 2014 [16], U-Net by Ronneberger 2015 [7]) as well as the developments in open source libraries and geoscience specific libraries, GPU enabled high-performance computing and cloud computing and the emergence of data analytics platforms.

Waldeland and Solberg [1] were the first to use supervised deep learning methods in seismic datasets. In their work, they used a sliding window approach where 65x65x65 cube patches were extracted from their seismic data and a 7 hidden-layer CNN was used to classify the center pixels of the overlapping patches over and over again to classify each pixel in their dataset as salt or not salt. In this paper, it was mentioned that one of the challenges of working with seismic data is the very large size of the data requiring a high amount of memory in the GPU which limits the size of their input in order to justify their choice of using small patches and a sliding window approach. They mentioned that training the model on a single labelled inline slice and using data augmentation techniques of random stretching (±20%), random flipping of x and y axis, and random rotations (±180° in x, y and ±15° in z) was sufficient to classify the rest of the dataset. The work of Waldeland then inspired the summer project of Charles Rutherford IIdstad in ConocoPhillips in 2017 who implemented the MalenoV [17] (machine learning of Voxels) software for multi facies classification on 3D seismic datasets which uses a very similar sliding window image segmentation method as the Waldeland's implementation. In their GitHub repository, it was mentioned that their classification time was the major problem of their sliding window approach. Then, their implementation on Matlab using Keras was translated into Python using Pytorch by Lukas Mosser. This report is available at [18] on Github and was taken as the starting point for the SWC approach we used in this project.

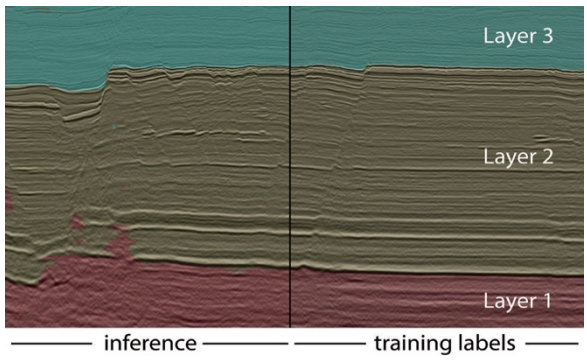

Figure 4: Classification on a 3D seismic dataset by Badrinarayan et al [11]. Right hand side shows the inline slice used for training, and the left side shows the predictions after training.

II. Methodology & Software Development Life Cycle (SDLC)

During the internship, a sliding window classifier (SWC), fully convolutional networks (FCNs), conditional random fields (CRFs) as well as 2D/3D visualization tools were implemented. For the majority of the project, agile software development strategies were used where small parts of the software were implemented, tested and then integrated. However, no formal unit tests are implemented, and individual parts were tested manually by inspection. Also, often there were no clear integration tests we could apply since there aren't well-defined input-output relationships. For this reason, the integration and functionality of the code were solely verified by the accuracy of their prediction and visual inspection of the results.

Everything is implemented in the Google-Colab environment using python 3. However, some libraries we use are implemented in other programming languages (such as the CRF implementation of Krahenbuhl which is implemented using C++). There are two main Jupyter notebooks (SWC_Notebook.ipynb & FCN_Notebook.ipynb) which do everything from training, predicting, CRF post-processing to visualization. Everything that is required to be installed (e.g. external libraries used, the CRF model) is open source and is installed within the notebooks. The only requirement for the user is to enable the GPU available on Google-Colab.

The SWC approach was developed as an extension to a pre-existing PyTorch implementation of the MalenoV software by Mosser [19]. This GitHub repository contained python files, Jupyter notebooks and data files. The most important parts contained in this repository were the train.py and test.py files which are used to train a model using few slices of labelled data and to make label predictions on different slices of the seismic cube. These predictions are saved into two different files of labels.npy and indices.npy which contains the labels and their indices on the seismic cube respectively. These two python files have dependencies in four different python files of datasets.py, model.py, options.py and utils.py which all have several classes and functions in them, but their main usages are to slice the 65x65x65 seismic cubes (patches) around the indices for training and predicting, define the model to be used for training, set the command-line options, and to save the checkpoints (per epoch) of the trained model respectively. Although this was a relatively recently written software some of the libraries imported were old and no longer existed and minor changes had to be done to make it work. For example, the shuffleSplit class imported from sklearn.cross_validation in several python files no longer existed and moved to sklearn.model_selection.

The internals of this class were also changed, and minor changes had to be done inside the python files. Also, some changes had been made to the original code to make it more functional. For example, reflective padding was added to the input dataset so that the entire seismic cube (including the pixels on the edges of the cube) can be predicted. We added the functionality to apply data augmentation to the extracted patches. We also added few functionalities to make the code easier to work with for example a function was added into train.py to automatically plot and save training and validation accuracy/loss as a function of number of epochs.

To use the SWC software described above, we created the notebook SWC_Notebook.ipynb which is used for everything from training to prediction to visualization. We load the SWC repository containing all the python files and datasets into SWC_Notebook.ipynb. Then we train the model using few inline slices of the dataset and make predictions on the entire seismic cube. We also load the CRF repository and apply the CRF model to the predicted cube in all three-slicing direction (i.e. we apply the CRF to inline slices, crossline slices and vertical slices of the predicted cube).

The FCN software, on the other hand, was developed as standalone software and makes use of the Keras library. In contrast to the SWC software as described above, everything is implemented in a single Jupyter notebook (FCN_Notebook.ipynb) which doesn't have any external dependencies apart from the usual python libraries that are imported. We believe this makes the software tidier and relatively easier to use and modify. Inside FCN notebook we have 4 different models of a 1 down-sampling autoencoder, a 2 down-sampling autoencoder and two U-Net models (of which one of them is size invariant and would work for any dataset). We use a small number of inlines and crosslines to train the selected model. Since the size of the training dataset (number of training images) for FCN is much smaller than that of the SWC (number of patches), we also use various data augmentations at the beginning (e.g. horizontal flips) to increase the size of the training dataset. During the training, we only save the model with the best validation accuracy and then use this model to predict the entire volume. Then, we apply the CRF model to our prediction the same way as in the SWC notebook.

We also have a notebook for model averaging (model_average.ipynb) where the predictions from various models can be loaded and averaged and the pixel-wise/IoU accuracy of the averaged prediction can be checked for a given test set. Finally, we have a 3D visualization notebook (3D_Visualize.ipynb) which makes use of the Mayavi library to make interactive 3D visualizations of the predicted cubes and seismic amplitudes (see appendix III).

	Size of input image n	Number of input channels	f	p	s	Size of output image (n+2p-f)/s+1	Number of output channels or filters	Number of output neurons	Size of Filter + 1	Number of Parameters
Conv1	65	1	5	2	4	17	50	245650	126	6300
Conv2	17	50	3	1	2	9	50	36450	1351	67550
Conv3	9	50	3	1	2	5	50	6250	1351	67550
Conv4	5	50	3	1	2	3	50	1350	1351	67550
Conv5	3	50	3	1	2	2	50	400	1351	67550
	Size of input							Number of output neurons		
FC1	400							50	401	20050
FC2	50							10	51	510
Softmax	10							5	11	55
							Total Neurons	290165	Total Parameters	297115

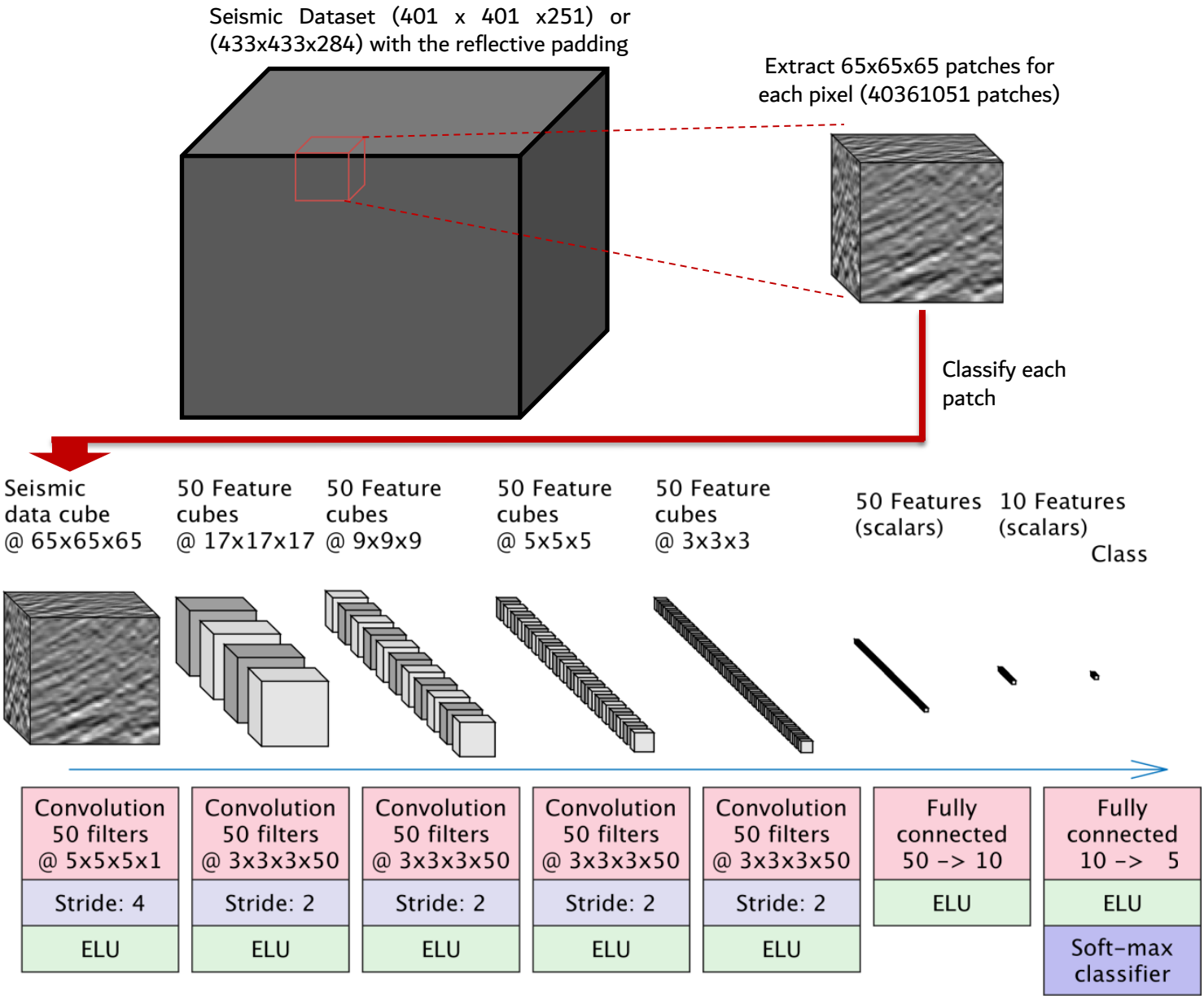

Figure 5: The CNN model used as the classifier of the SWC approach is shown in the table. Batch normalization was used between each convolutional layer as well as a dropout rate of 0.2. The model used for the Salt dataset is identical to the above but the final SoftMax layer has 2 instead of 5 output neurons. The bottom figure shows the design diagram for the SWC approach. The Table of parameters for the FCN model are attached in the appendix. The design diagram for the FCN model is very similar to figure 2 but with the parameters in the appendix. For the FCN, we feed in the in-line or crossline slices as the input and then stack the 2D segmentation maps to obtain the predicted cube.

III. Results

The Forties dataset contains five different classes corresponding to five different geological units of Overburden, Sele, Forties, Lista, and Underburden. These classes correspond to labels 0, 1, 2, 3 and 4 in figure 6 respectively.

We train our FCN models using only four inline slices (of index 0, 150, 300, 400, out of 401 inlines) and four crossline slices (of index 0, 150, 300, 400, out of 401 crosslines). We also use one inline slice (of index 225) and one crossline (of index 75) as our validation set. We apply data augmentation technique of only horizontal flips to the training set to increase its size by two-fold. As the loss function, we use categorical cross-entropy and we weight the loss for every class proportionally to their reciprocal proportion in the training dataset. This was done since it was observed that classes with lower abundance (e.g. class 1) were being predicted much less accurately than classes with very high abundance (e.g. classes 0 and 4). We use the Adams algorithm as the optimiser algorithm which was observed to get better accuracies than the standard SGD algorithm. Then, we train the model for 150 epochs using 10 batches, each containing four images (X) and their corresponding segmentation maps (y). The training time of the FCN models are generally very fast and takes around ten minutes for the U-Net models that contain more than five million parameters. During training, no data augmentation techniques are used, and it was observed that many usages of data augmentation techniques (e.g. rotations by small degrees, adding random noise, random zooms, etc.) resulted in lower training and validation accuracies. Only the model that gives the highest validation accuracy during training is saved (i.e. we don't save the model for every epoch). Then, once the model is trained, we apply the model for all the inline slices to get the prediction for the entire seismic cube. Then, we horizontally flip the seismic amplitudes, predict the cube by applying the model for every inline slice and horizontally flip our prediction back into the original orientation. Finally, we predict the cube by applying our model on the crossline slices instead of the inline slices. Now, that we have three predicted cubes that are roughly equivalent in terms of their accuracy, we model average them (by simply taking the mode) to get a prediction which is generally more accurate (in terms of both pixel-wise and IoU accuracy) than any individual prediction. The prediction time of the

FCN approach is very fast, and it takes 15 seconds to predict the entire seismic volume.

We train the CNN of our SWC approach using only eight inline slices (of index 0, 60, 120, 180, 240, 300, 350, 400 out of 401 inline slices). Then we extract 3D cube patches for each pixel as shown in figure 5. This dataset is split into training and validation sets with ratio 0.8 and 0.2 respectively. We use cross-entropy as our loss function and Adams as our optimization algorithm. As with the FCN, we weight the loss for every class proportionally to their reciprocal proportion in the training dataset. The CNN model is trained for 10 epochs using 30% of the available training and 10% of the available validation datasets per epoch. Training and validation sets are shuffled at the end of each epoch so that we feed in a different training and validation sets for the next epoch. In datasets.py we have a functionality to add 3D data augmentation of 90 degrees rotations in random directions. However, it was observed that adding data augmentations do not improve the results and for this reason, no data augmentation is used during the training. Also, there aren't many publicly available libraries for 3d data augmentations and therefore the user may have to manually implement their data augmentations if they desire to do so (we only provided a simple example). However, we believe that data augmentation is generally not necessary for a sliding window classifier as it already generates a massive training dataset by extracting patches and we didn't observe any overfitting problems as displayed in the training curve of the SWC in figure 6. The training time for the SWC approach takes around an hour which is slightly more than that of the FCN approach. Once the model is trained, we extract patches for the entire seismic volume and feed this to our model in order to predict the entire seismic volume. However, this requires a massive amount of memory on the GPU which we don't have on Google-Colab. Therefore, we extract patches for each inline slice and segment the seismic volume slice by slice. Then, we use a script to combine all the indices and labels files and form the 3D seismic cube segmentation map. The prediction time of the SWC approach is very slow compared to the FCN approach and it takes around 13 hours to predict the entire seismic volume (2 minutes per inline slice). The results from the SWC/FCN approaches are displayed in figure 6.

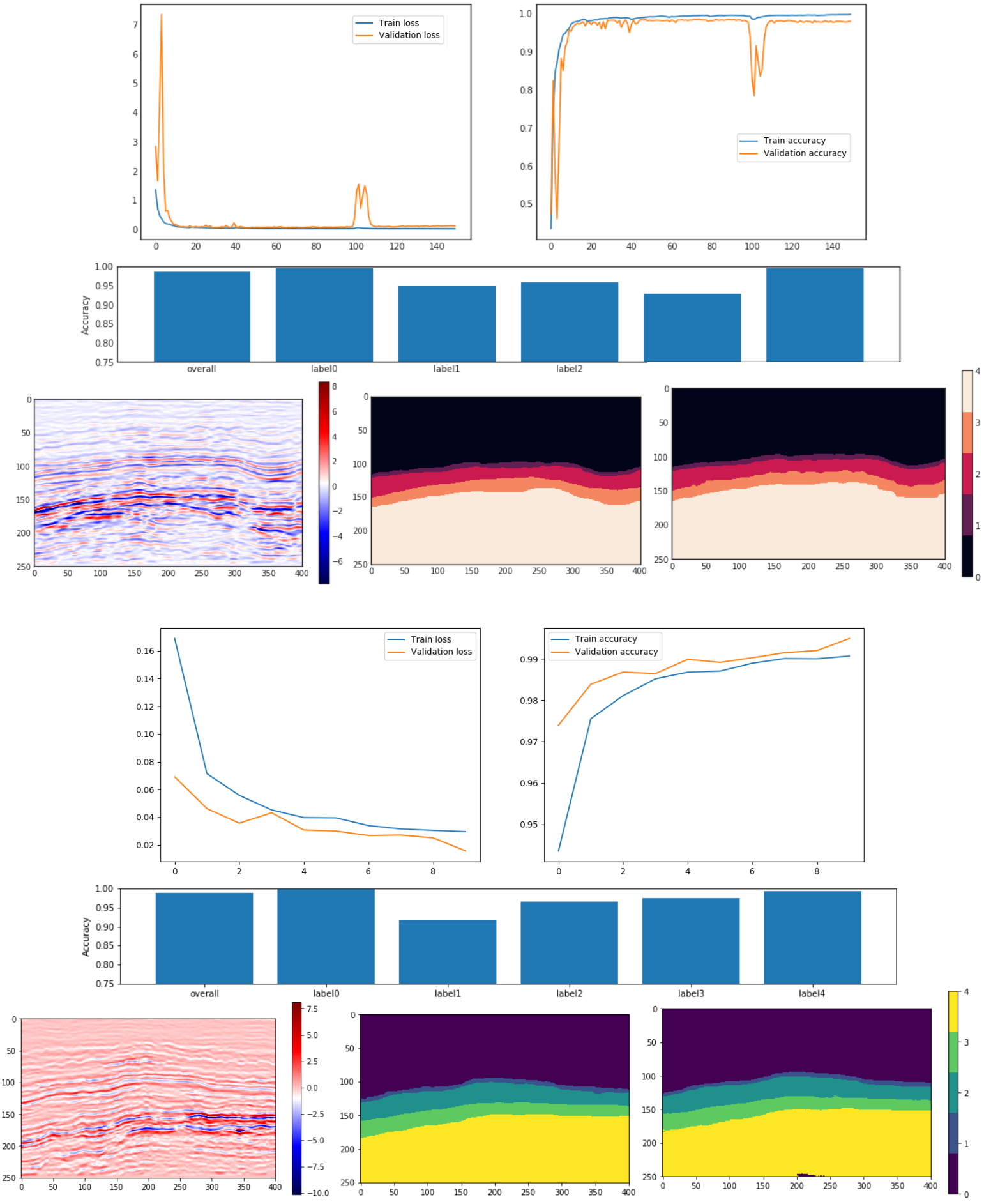

Figure 6: The top figure shows the training and validation accuracy and loss for the U-Net model. The second figure shows the accuracy of each class predicted for the entire seismic cube where the overall pixel-wise accuracy is 0.9870 and the IoU accuracy is 0.9345. The third figure shows the seismic amplitudes for the 330th crossline, the ground truth labels, and the FCN predicted labels respectively. The bottom three figures are for the SWC approach. The overall pixel-wise accuracy on the SWC predicted seismic cube is 0.9894 and the IoU accuracy is 0.9426. The bottom figure is for the 76th inline.

In the accuracy figure of the FCN approach (first plot in figure 6) it was observed that the U-Net model slightly overfitted the data, where training accuracy of 99.5%, validation accuracy of 98.5% and test accuracy of 98.7% was obtained. This was not surprising as the size of the training data was very small (only 8 inline slices) and excessive amounts of data augmentations were not used. When the size of the training dataset was raised from 8 to 80 inline slices, it was observed that the model no longer overfitted the data and test accuracies of around 99.5% were observed. We didn't use excessive amounts of data augmentation because many of the data augmentation techniques that were tested (e.g. vertical flips, random Gaussian noise, random zoom, random rotations of less than 5 degrees, etc.) resulted in lower training and validation accuracies. Only horizontal flips which increases the training size by two-fold proved to be a useful data augmentation to improve the validation accuracy.

The sliding window classifier, on the other hand, seems to neither overfit nor underfit the training dataset where similar training and validation accuracies are observed. For both FCN and SWC, we only used 8 inline slices for training but since the SWC extracts patches around each pixel, it had a much larger training dataset. This larger training dataset is probably the reason for not overfitting the data with the SWC approach.

Once the seismic cube segmentation maps are obtained for the seismic cube by the FCN/SWC approach, we apply the CRF model to the segmentation map in all three slicing directions (inline, crossline, vertical) as shown in figure 7. For our Gaussian kernel (equation 6), we set the values of $\theta = 2$ and $w = 6$. These values were obtained by visually inspecting the output of the CRF model as displayed in figure 3. It was observed that when the predicted segmented cubes have low pixel-wise/IoU accuracy, the CRF model improved the overall accuracy significantly whereas when the prediction accuracy was high the CRF model improved the overall accuracy only slightly. This is demonstrated in figure 7.

Our best prediction on the Forties dataset was obtained using an ensemble approach where we model average (by taking the mode) our three best models (the size invariant U-Net, size variant U-Net and the SWC approach). For these approaches, we originally obtained pixel-wise/IoU accuracies of 0.9870/0.9345, 0.9856/0.9276, 0.9894/0.9426 respectively. After model averaging the accuracy was improved to 0.9902/0.9441. Then, we applied the CRF model to the ensemble prediction and the accuracy improved only very slightly to 0.9903/0.9442. Our ensemble approach is displayed in figure 8.

For the salt dataset, the same methods (e.g. the size invariant U-Net and the CNN model in figure 5) were used. For the FCN/SWC approach, similar accuracies were obtained as the Forties dataset. Some example plots from training and some example predictions for this dataset are provided in the appendix.

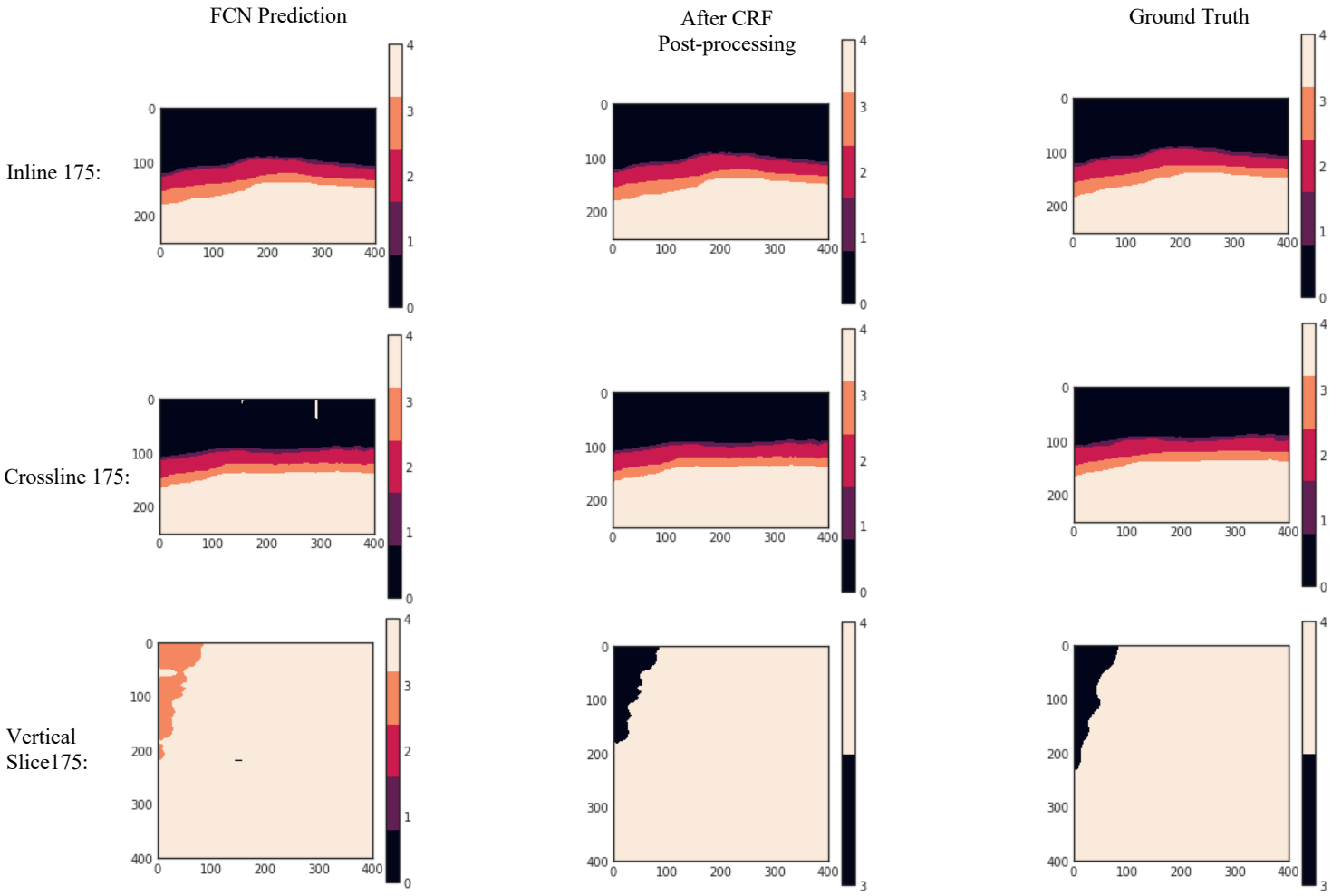

Figure 7: Applying the CRF model to the 175th inline, crossline, and vertical slices respectively. When applying CRF to the segmented cube, we first apply it across in-lines, then across crosslines and finally across the vertical slices. From the top figure it can be seen that when the predictions are already very high accuracy CRF model doesn't help much. The second and third column shows that the CRF model can improve the accuracy by correctly classifying some of the wrongly classified parts. However, the third column also shows that sometimes we can misclassify some parts that were initially correctly classified (e.g. the class 3 cluster at around y = 200).

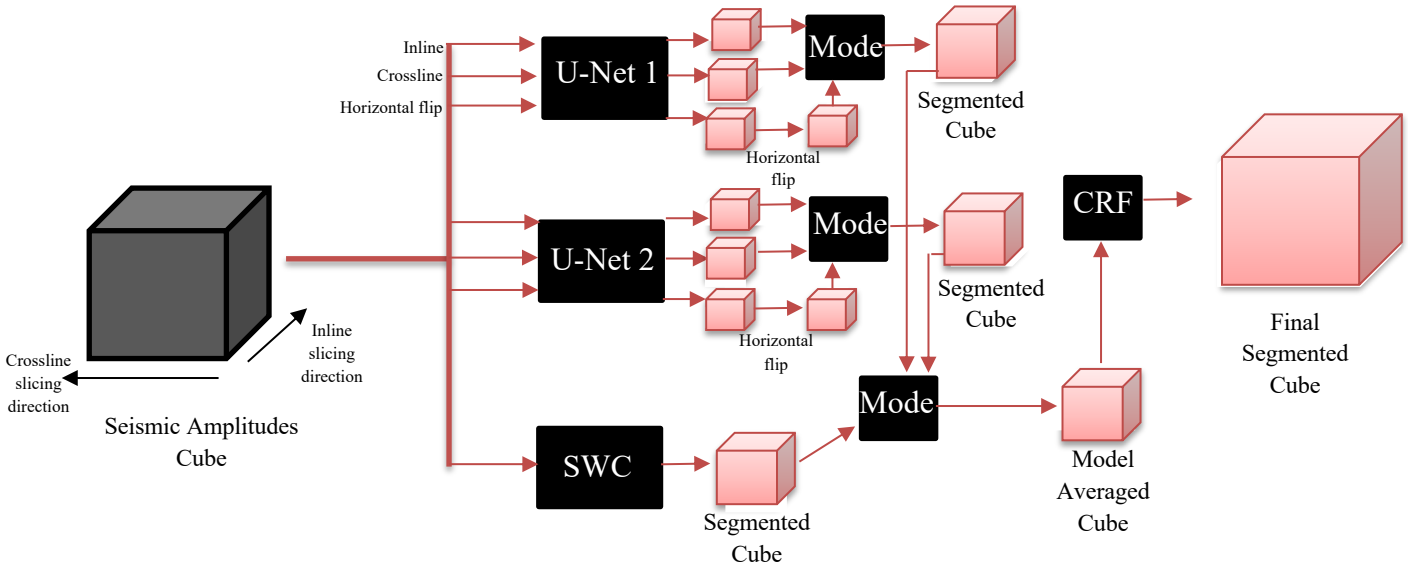

Figure 8: Ensemble approach used. We feed the original amplitudes cube to the U-Net models (model 3 and model 4 in our FCN_Notebook.ipynb) as inline slices, crossline slices and after horizontal flipping. We get three predictions for the cube segmentation map. We model average (by taking the mode) those predictions to get a better cube segmentation map. For the SWC, we feed in the patches extracted around each pixel to obtain a cube segmentation map. We model average (by taking the mode) the three segmentation maps obtained from U-Net1, U-Net2 and SWC. Then, we apply the CRF model on this model averaged segmented cube.

IV. Discussion

The performance in terms of prediction time of our FCN and SWC approaches can be estimated by only considering the number of neurons in our models and the number of patches that has to be fed to the models. Here the number of neurons is a rough estimate of the computational cost to apply the model to a given patch and the number of patches is directly proportional to the prediction time. For the FCN approach, we used the U-Net model (see appendix) which contained 5.6 million neurons and required 401 patches (401 inline slices of size 401x251) to predict the entire seismic cube. The CNN model (see the table in figure 5) of the SWC approach, on the other hand, had 0.29 million neurons and required 40361051 patches (the number of pixels in the seismic volume 401x401x251) to predict the entrire volume. We expect that the prediction time of each approach can be estimated as the number of neurons (N) multiplied by the number of patches (M) multiplied by a scaling constant (α) that depends on the environment e.g. performance of the GPU available / efficiency of the libraries used and etc. With this estimate, we expect that the FCN approach will be X times faster than the SWC approach given by,

$$X \sim \frac{\alpha_{SWC} \times N_{SWC} \times M_{SWC}}{\alpha_{FCN} \times N_{FCN} \times M_{FCN}}. \quad (7)$$

Plugging in our numbers for N and M of the FCN and SWC approaches (for the forties dataset), we obtain $X \sim 5212 \frac{\alpha_{SWC}}{\alpha_{FCN}} \sim 5000.$ Here we assume that the environment term ($\frac{\alpha_{SWC}}{\alpha_{FCN}}$) is around 1 as we run both the FCN and SWC approaches on Google-Colab using the same GPU available. The only significant difference in the environment is that the SWC approach makes use PyTorch library which is based on Torch whereas the FCN approach makes use of the Keras library which is based on TensorFlow. Since both Torch and TensorFlow are very popular libraries for machine learning tasks, a massive performance difference between the two libraries is not expected.

In our experiments, it was observed that the FCN approach was indeed the significantly faster approach where the U-Net model took around 15 seconds to predict the entire seismic volume (401 inline slices of size 401x251). Comparing this to the SWC approach which took around 13 hours to predict our seismic volume, we see that the prediction time of the FCN approach was about 3000 times faster than that the SWC approach. This number is very close to our estimated number of 5000 obtained from equation 7. Here we see that estimating the

computational cost of the algorithm as the number of neurons multiplied by the number of patches does seem to be a good estimate where our estimate for X seems to be of the correct order of magnitude. The difference between our estimate and the actual value is very sensible and can easily be justified by the environment factor and the fact that the FCNs only have convolutional layer neurons whereas CNNs also have fully connected layer neurons.

Despite being a much more computationally expensive approach, the SWC seems to have a slight edge over the FCN approach in terms of prediction accuracy. This seems to be the case, at least on the Forties dataset (Figure 6), where the SWC approach obtains slightly higher validation and test accuracies (in terms of both pixel-wise and IoU accuracies) than the FCN. We believe that this difference is mainly due to the larger size of the training dataset SWC has due to its ability to extract patches for each pixel. This results in less overfitting to the training data and, therefore, higher validation accuracies can be obtained. Another major difference between SWC and FCN that may have an impact on accuracy is that the SWC approach has 3D information (due to cube patches), whereas the FCN approach has depth information (due to 2D slice patches) when making predictions. The CNN of the SWC approach can see 3D seismic amplitudes around each pixel being predicted which probably helps to classify each pixel more accurately than having 2D patches (as in the case of the FCN approach). However, despite having 3D information, the SWC approach doesn't have information about the location of the pixels being predicted and this information (especially the depth of the seismic dataset) seems to be a very important feature for our dataset. The consequences of not having depth in the training dataset can be seen clearly on the SWC prediction on figure 6 where a small region of Underburden (label 4) is misclassified as Overburden (label 0). The importance of depth in this dataset was also come across when training the FCN models. Initially, we used to have horizontal and vertical flip data augmentations when training the U-Net model and this approach gave us pixel-wise validation accuracies of around 98.3%. Visual inspections of predictions revealed that similar misclassification of the Overburden and Underburden classes were present in the FCN approach (an example is the crossline 175 prediction in figure 7). Then, it was realised that the difference between the Overburden and the Underburden classes could be learned much more effectively when the

vertical flip data augmentations were removed. After the removal of vertical flips, validation accuracies of up to 98.7% were obtained. This was interpreted to be a result of vertical flips not allowing the network to use the depth (y-axis coordinate) of a pixel as an attribute to discriminate between Overburden and Underburden since the vertical flips used to make the Underburden layer on top of the Overburden layer. Overall, depth seems to be a more important feature than having 3D seismic amplitudes in this dataset where the pixel-wise training accuracies of the U-Net model can go up to 99.7% whereas the SWC training accuracies goes as high as 99.0%. Also, it was observed that using more training dataset (e.g. 80 inline/cross-line slices instead of 8 inline/crossline slices) the FCN gets higher accuracies than the SWC approach with pixel-wise validation and testing accuracies of around 99.5%. Nevertheless, SWC approach is the more accurate approach when dealing with small training datasets.

For the SWC approach, one of our initial expectations was that the padding used for predicting the outer pixels could cause some trouble and that the outer pixels of the cube that are less than half cube patch size from the cube edge could be predicted worse than the rest of the pixels. However, we did not come across such a problem which suggests that the reflective padding used worked very well for this seismic dataset. A possible reason for this may be that even at the very edges of the cube, at least half of the cube patches are from the original unpadded dataset, and for this reason, the padding does not significantly affect the outcome of CNN. Alternatively, it may be that since we also used some cube patches that had padding during the training, the CNN model learned to handle this well.

The model averaging approach didn't prove to be too useful. We model-averaged our two U-Net models (size variant and size invariant U-Nets) with the SWC classifier prediction. U-Net models already had pixel-wise accuracies of 98.70% and 98.56%, and IoU accuracies of 93.45% and 92.76% respectively, and the SWC had pixel-wise accuracy of 98.94% and IoU accuracy of 94.26%. The resultant averaged model (by taking the mode) had a pixel-wise/IoU accuracies of 99.02% and 94.51% respectively. Applying CRF postprocessing to this didn't make any significant difference and the pixel-wise/IoU accuracies improved to 99.03% and 94.52% respectively. We suspect that the reason model averaging did not improve the accuracy

significantly could be that the U-Net predictions were a bit interrelated. Model averaging usually works best when the predictions come from completely different methods. For this reason, it would be interesting to implement a third method such as a GAN or ResNet-50 and see if model averaging this method with the U-Net and the SWC would result in a prediction with much higher accuracy.

V. Conclusion

There are many possible future works that can be done as an extension to this project. If time permitted, we would be interested to investigate CRF-RNNs [19] where the CRF modelling is fully integrated with the DNNs instead of being used as post-processing tools. This allows the whole network to be trained end to end with the standard back-propagation algorithm. In various papers such as [19, 20] it was mentioned that CRF-RNNs achieve better accuracy scores compared to deep learning algorithms followed by CRF post-processing. It would be interesting to see if this would be the case on seismic interpretation tasks. Another interesting direction would be to use adversarial training approaches instead of using CRF models to learn the structure in the data. This approach was first applied to semantic segmentation tasks by Luc et al. [21] in 2016, and they mention that their approach resulted in better accuracy results than using FCN followed by CRF as well as using CRF-RNN approaches.

To conclude, we applied sliding window classifiers and fully convolutional networks to seismic datasets. From our experiments, we observed that the U-Net model we used as our FCN approach has achieved higher performance in terms of prediction accuracy and prediction/training time than the SWC approach. However, we recognised that the SWC approach has an advantage in training when small training datasets are used. This ability of SWC to extract patches and generate large training datasets enables it to overfit significantly less to the training data. For this reason, we observed that when the models were only trained with 8 inline slices, the SWC approach achieved higher validation accuracies (despite lower training accuracies) than the FCN. The CRF post-processing was observed to be very promising when prediction accuracies were low where the overall pixel-wise accuracies could be improved significantly. However, for high accuracy prediction (e.g. pixel-wise accuracies > 98.5%), the CRF model often only barely improve the overall accuracy (by often < 0.1%).

VI. Code Metadata

Nr.	Code metadata description	Please fill in this column
C1	Current code version	None
C2	Permanent link to code/repository used for this code version	$https://github.com/TayfunKaraderi/ACSE-9-Project-Automatic-Seismic-Interpretation-Techniques$
C3	Legal Code License	MIT Licence
C4	Code versioning system used	None
C5	Software code languages, tools, and services used	Python 3
C6	Compilation requirements, operating environments & dependencies	Google-Colab
C7	If available Link to developer documentation/manual	$https://github.com/TayfunKaraderi/ACSE-9-Project-Automatic-Seismic-Interpretation-Techniques$
C8	Support email for questions	karaderitayfun@gmail.com

Table 1: Code Metadata

VII. References

[1] Waldeland, A.U., Solberg, A.H.S.S. [2017]. Salt Classification Using Deep Learning. Conference: 79th EAGE Conference. 10.3997/2214- 4609.201700918.

[2] Sabbione, J., Velis, D. [2010]. Automatic first-breaks picking: New strategies and algorithms. Geophysics. **75**. 10.1190/1.3463703.

[3] Yu, Y. [2011]. Automatic Horizon Picking in 3D Seismic Data Using Optical Filters and Minimum Spanning Tree. SEG Technical Program Expanded Abstracts. **30**.

[4] Admasu, F., Tönnies, K. [2005]. An Approach towards Automated Fault Interpretations in Seismic Data. Conference: Simulation und Visualisierung. 207-220.

[5] Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. Master's thesis, 1970.

[6] Ciresan D, Giusti A, Gambardella LM, Schmidhuber J. Deep neural networks segment neuronal membranes in electron microscopy images In: Pereira F, Burges CJC, Bottou L, Weinberger KQ, editors. Advances in Neural Information Processing Systems 25. Curran Associates, Inc.: 2012. p. 2843–851.

[7] Ronneberger, O., Fischer P., Brox, T., U-Net: Convolutional Networks for Biomedical Image Segmentation, arXiv preprint arXiv:1505.04597**.**

[8] Krahenbuhl P and Vladlen Koltun, [2012], Efficient Inference in Fully Connected CRFs with Gaussian edge potentials. In *NIPs,* pages 109-117.

[9] Github repository for CRF Inference algorithm by Lucasb-eyer: https://github.com/lucasb-eyer/pydensecrf, last accessed on 27/08/19.

[10] Zhao, T., Jayaram, V., Roy, A. and Marfurt, K.J. [2015]. A comparison of classification techniques for seismic facies recognition. Interpretation, **3**(4), SAE29-SAE58.

[11] Qi, J., Lin, T., Zhao, T., Li, F. and Marfurt, K. [2016]. Semisupervised multiattribute seismic facies analysis. Interpretation, **4**(1), SB91 SB106.

[12] Meldahl, P., Heggland, R., Bril, B. and de Groot, P. [2001]. Identifying faults and gas chimneys using multiattributes and neural networks. The Leading Edge, **20**(5), 474-482.

[13] Waldeland, A.U., Jensen, A.C., Gelius, L.J. and Solberg, A.H.S. [2018]. Convolutional neural networks for automated seismic interpretation. The Leading Edge, **37**(7), 529-537.

[14] Badrinarayanan, V., Kendall, A. and Cipolla, R. (2015). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. arXiv preprint, arXiv:1511.00561.

[15] Krizhevsky, A., Sutskever, I. and Hinton, G.E. [2012]. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, 1097-1105.

[16] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S. and Bengio, Y. [2014]. Generative adversarial nets. Advances in neural information processing systems, 2672-2680.

[17] Github repository for MalenoV by Peter Bolgebrygg: https://github.com/bolgebrygg/MalenoV, last accessed on 25/06/19.

[18] Github repository for MalenoV by Lukas Mosser: https://github.com/LukasMosser/asi-pytorch, last accessed on 25/06/19.

[19] Zheng S., Jayasumana S, et al. Conditional random fields as Recurrent Neural Networks, [2015], In *Proceedings of IEEE International Conference on Computer Vision,* pages 1529-1537.

[20] Arnab A., Zheng S., [2018], Conditional Random Fields Meet Deep Neural Networks for Semantic Segmentation. IEEE Signal Proc. **35**, 37-52.

[21] P. Luc, C. Couprie, S. Chintala, and J. Verbeek. [2016], Semantic segmentation using adversarial networks. In NIPS Workshop on Adversarial Training.

Appendix I: Salt Dataset

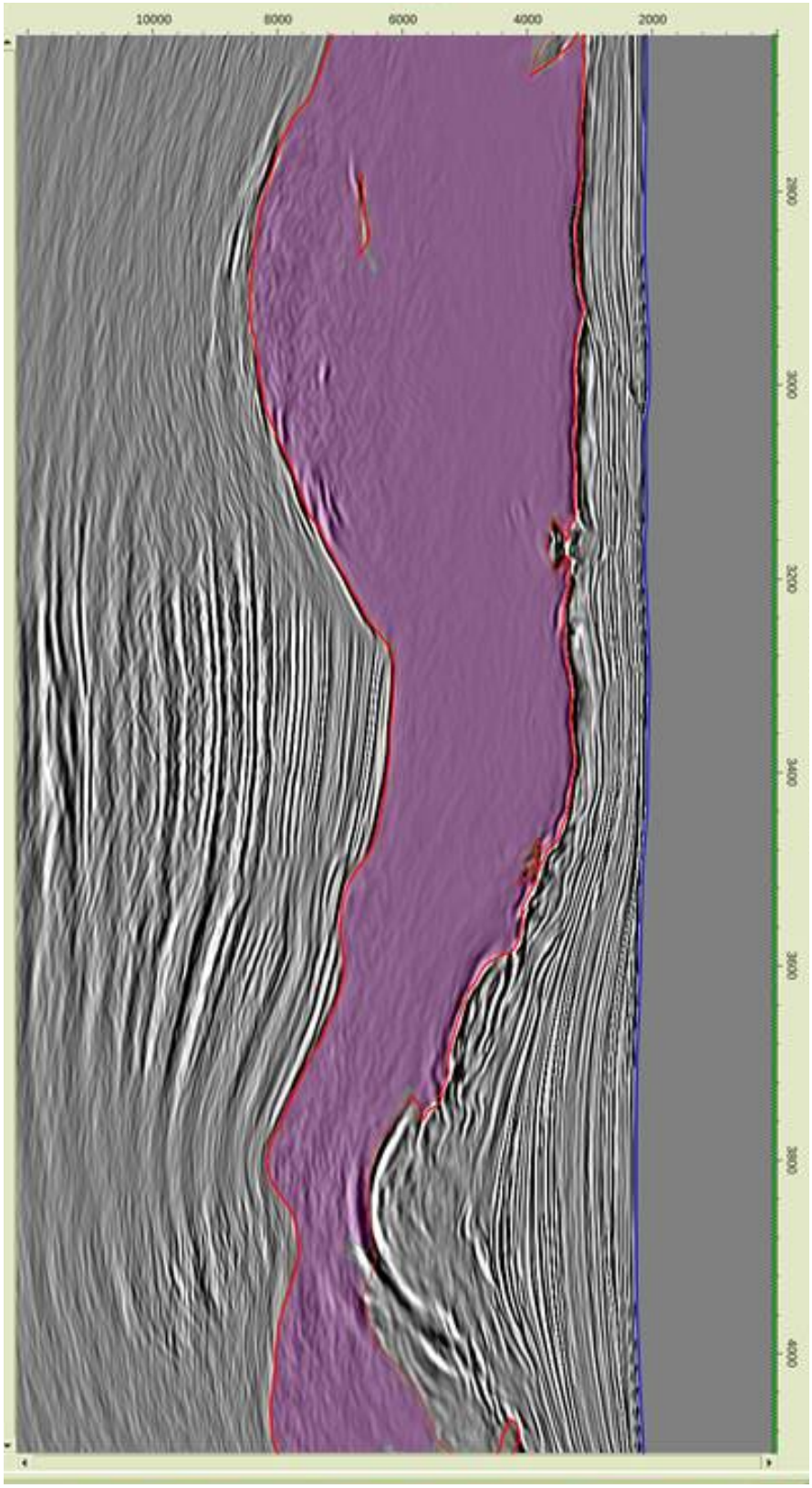

Figure: The size invariant U-Net model (model 4 in FCN_Notebook.ipynb) was trained for the salt dataset using 8 inline slices. Overall validation and test accuracies (pixel-wise) of around 99% were obtained. The figure shows our prediction of salt (in purple) and not salt (in white) for one of the test inline slices. The red lines are the original horizons.

Appendix II: U-Net Model Parameters

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 401, 251, 1)	0	
conv2d_1 (Conv2D)	(None, 401, 251, 12)	312	input_1[0][0]
batch_normalization_1 (BatchNor	(None, 401, 251, 12)	48	conv2d_1[0][0]
dropout_1 (Dropout)	(None, 401, 251, 12)	0	batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, 401, 251, 12)	3612	dropout_1[0][0]
batch_normalization_2 (BatchNor	(None, 401, 251, 12)	48	conv2d_2[0][0]
dropout_2 (Dropout)	(None, 401, 251, 12)	0	batch_normalization_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 200, 125, 12)	0	dropout_2[0][0]
zero_padding2d_1 (ZeroPadding2D	(None, 204, 131, 12)	0	max_pooling2d_1[0][0]
conv2d_3 (Conv2D)	(None, 200, 126, 24)	8664	zero_padding2d_1[0][0]
batch_normalization_3 (BatchNor	(None, 200, 126, 24)	96	conv2d_3[0][0]
dropout_3 (Dropout)	(None, 200, 126, 24)	0	batch_normalization_3[0][0]
conv2d_4 (Conv2D)	(None, 200, 126, 24)	14424	dropout_3[0][0]
batch_normalization_4 (BatchNor	(None, 200, 126, 24)	96	conv2d_4[0][0]
dropout_4 (Dropout)	(None, 200, 126, 24)	0	batch_normalization_4[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 100, 63, 24)	0	dropout_4[0][0]
zero_padding2d_2 (ZeroPadding2D	(None, 104, 69, 24)	0	max_pooling2d_2[0][0]
conv2d_5 (Conv2D)	(None, 100, 64, 36)	25956	zero_padding2d_2[0][0]
batch_normalization_5 (BatchNor	(None, 100, 64, 36)	144	conv2d_5[0][0]
dropout_5 (Dropout)	(None, 100, 64, 36)	0	batch_normalization_5[0][0]
conv2d_6 (Conv2D)	(None, 100, 64, 36)	32436	dropout_5[0][0]
batch_normalization_6 (BatchNor	(None, 100, 64, 36)	144	conv2d_6[0][0]
dropout_6 (Dropout)	(None, 100, 64, 36)	0	batch_normalization_6[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 50, 32, 36)	0	dropout_6[0][0]
conv2d_7 (Conv2D)	(None, 50, 32, 48)	43248	max_pooling2d_3[0][0]
batch_normalization_7 (BatchNor	(None, 50, 32, 48)	192	conv2d_7[0][0]
dropout_7 (Dropout)	(None, 50, 32, 48)	0	batch_normalization_7[0][0]
conv2d_8 (Conv2D)	(None, 50, 32, 48)	57648	dropout_7[0][0]
batch_normalization_8 (BatchNor	(None, 50, 32, 48)	192	conv2d_8[0][0]
dropout_8 (Dropout)	(None, 50, 32, 48)	0	batch_normalization_8[0][0]

max_pooling2d_4 (MaxPooling2D)	(None, 25, 16, 48)	0	dropout_8[0][0]
conv2d_9 (Conv2D)	(None, 25, 16, 384)	461184	max_pooling2d_4[0][0]
batch_normalization_9 (BatchNor	(None, 25, 16, 384)	1536	conv2d_9[0][0]
dropout_9 (Dropout)	(None, 25, 16, 384)	0	batch_normalization_9[0][0]
conv2d_10 (Conv2D)	(None, 25, 16, 384)	3686784	dropout_9[0][0]
batch_normalization_10 (BatchNo	(None, 25, 16, 384)	1536	conv2d_10[0][0]
dropout_10 (Dropout)	(None, 25, 16, 384)	0	batch_normalization_10[0][0]
conv2d_transpose_1 (Conv2DTrans	(None, 50, 32, 96)	331872	dropout_10[0][0]
concatenate_1 (Concatenate)	(None, 50, 32, 144)	0	conv2d_transpose_1[0][0] conv2d_8[0][0]
conv2d_11 (Conv2D)	(None, 50, 32, 96)	345696	concatenate_1[0][0]
batch_normalization_11 (BatchNo	(None, 50, 32, 96)	384	conv2d_11[0][0]
dropout_11 (Dropout)	(None, 50, 32, 96)	0	batch_normalization_11[0][0]
conv2d_12 (Conv2D)	(None, 50, 32, 96)	230496	dropout_11[0][0]
batch_normalization_12 (BatchNo	(None, 50, 32, 96)	384	conv2d_12[0][0]
dropout_12 (Dropout)	(None, 50, 32, 96)	0	batch_normalization_12[0][0]
conv2d_transpose_2 (Conv2DTrans	(None, 100, 64, 84)	72660	dropout_12[0][0]
concatenate_2 (Concatenate)	(None, 100, 64, 120)	0	conv2d_transpose_2[0][0] conv2d_6[0][0]
conv2d_13 (Conv2D)	(None, 100, 64, 48)	144048	concatenate_2[0][0]
batch_normalization_13 (BatchNo	(None, 100, 64, 48)	192	conv2d_13[0][0]
dropout_13 (Dropout)	(None, 100, 64, 48)	0	batch_normalization_13[0][0]
conv2d_14 (Conv2D)	(None, 100, 64, 48)	57648	dropout_13[0][0]
batch_normalization_14 (BatchNo	(None, 100, 64, 48)	192	conv2d_14[0][0]
dropout_14 (Dropout)	(None, 100, 64, 48)	0	batch_normalization_14[0][0]
conv2d_transpose_3 (Conv2DTrans	(None, 200, 128, 72)	31176	dropout_14[0][0]
zero_padding2d_3 (ZeroPadding2D	(None, 200, 128, 24)	0	conv2d_4[0][0]
concatenate_3 (Concatenate)	(None, 200, 128, 96)	0	conv2d_transpose_3[0][0] zero_padding2d_3[0][0]
conv2d_15 (Conv2D)	(None, 200, 128, 24)	57624	concatenate_3[0][0]
batch_normalization_15 (BatchNo	(None, 200, 128, 24)	96	conv2d_15[0][0]
dropout_15 (Dropout)	(None, 200, 128, 24)	0	batch_normalization_15[0][0]
conv2d_16 (Conv2D)	(None, 200, 128, 24)	14424	dropout_15[0][0]
batch_normalization_16 (BatchNo	(None, 200, 128, 24)	96	conv2d_16[0][0]

dropout_16 (Dropout)	(None, 200, 128, 24)	0	batch_normalization_16[0][0]
cropping2d_1 (Cropping2D)	(None, 400, 250, 12)	0	conv2d_2[0][0]
conv2d_transpose_4 (Conv2DTrans	(None, 400, 256, 12)	2604	dropout_16[0][0]
zero_padding2d_4 (ZeroPadding2D	(None, 400, 256, 12)	0	cropping2d_1[0][0]
concatenate_4 (Concatenate)	(None, 400, 256, 24)	0	conv2d_transpose_4[0][0] zero_padding2d_4[0][0]
zero_padding2d_5 (ZeroPadding2D	(None, 408, 258, 24)	0	concatenate_4[0][0]
conv2d_17 (Conv2D)	(None, 404, 254, 12)	7212	zero_padding2d_5[0][0]
batch_normalization_17 (BatchNo	(None, 404, 254, 12)	48	conv2d_17[0][0]
dropout_17 (Dropout)	(None, 404, 254, 12)	0	batch_normalization_17[0][0]
conv2d_18 (Conv2D)	(None, 401, 251, 12)	2316	dropout_17[0][0]
conv2d_19 (Conv2D)	(None, 401, 251, 5)	65	conv2d_18[0][0]

Total params: 5,637,533
Trainable params: 5,634,821
Non-trainable params: 2,712

Table: The U-Net model (model 3 in FCN_Notebook.ipynb) parameters is displayed.

Appendix III: 3D Visualisation

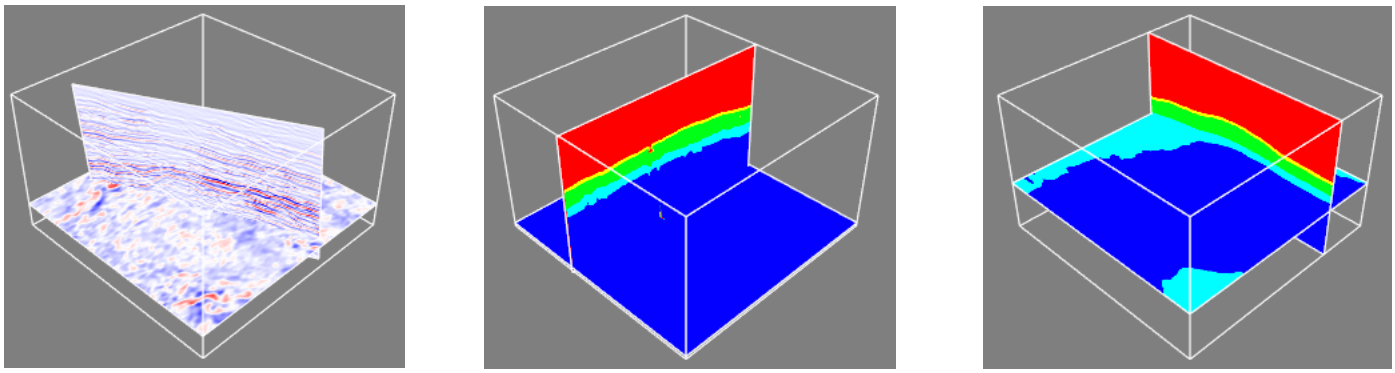

Figure: Shows examples of interactive visualisations for seismic cubes obtained from the 3D_Visualize.ipynb. The left cube is the amplitudes cube. The two cubes on the right display the segmentation map predicted from a U-Net model.