Imperial College London
Department of Earth Science and Engineering

Independent Research Project
Final Report

Machine Learning-based Classification of Europa's Fractures

by
Longyun Liao

longyun.liao19@imperial.ac.uk

Github login: acse-ll2819

Supervised by Dr. Adriana Paluszny

Submitted for the degree of
MSc in Applied Computational Science and Engineering

August 2020

Abstract

Various of machine learning-based models are developed in this project to extract features of the images of Europa fractures, including a basic autoencoder, a denoising autoencoder, a sparse autoencoder, a Restricted Boltzmann Machine (RBM), and stacked Restricted Boltzmann Machines. Then those features are grouped to find the repeating patterns of fractures on Europa. Despite classifying the fractures in an unsupervised way, semantic segmentation models to label the Europa images at pixel level are also explored. **This project is the first attempt in the literature to learn features of fractures on Europa based on machine learning strategies.**

Keywords: Europa, unsupervised-learning, semantic segmentation

Acknowledgments

Above all, I would like to express my appreciation to my Supervisor Dr. Adriana Paluszny, who presents patience and great support to me throughout this project. I would also like to appreciate Dr. Pappalardo who provides me with high-quality images from NASA and a fascinating opportunity to explore Europa. Finally, I would like to thank my teammate Ke Wen for discussing machine learning methods with me.

Contents

1. Introduction

Europa, Jupiter's moon, is considered to be one of the most promising places to explore extraterrestrial life. There are six spacecraft that have explored Europa to date. Those spacecraft missions of NASA captured thousands of images of Europa's surface and revealed the potential evidence suggesting the existence of liquid water below the surface, either today or at some time in the past.

Nowadays, the patterns of fractures on Europa are observed by human eyes (Leonard, Pappalardo and Yin, 2018). Different from previous work on Europa, we tried to find the patterns of fractures by machine learning strategies. The work of our project is divided into three major parts: data-preprocessing, unsupervised learning for classification, and semantic segmentation. The resolution of providing images from NASA'S JPL (Jet Propulsion Laboratory) ranges from 20 km/pixel to 6 m/pixel and these images with varying sizes map different areas of Europa. We applied unsupervised-learning methods on small images, images that probably contain only one type of fractures, to classify different types of fractures. And we applied semantic segmentation technique on larger images which contain more than one type of fractures, to segment different types of fractures in each image.

2. Background Research

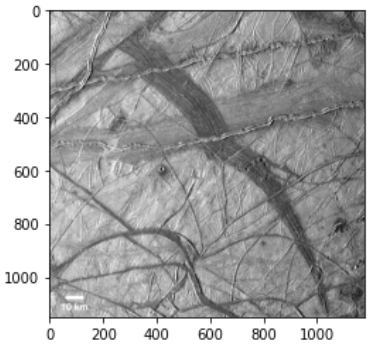
2.1 Data-preprocessing:

The images from JPL are not well-organized if regarded as a machine-learning training data set. Lots of images have relatively large white/black regions and text, which disturbs the training process. Also, the images were taken under different lighting so that optimizing the brightness over the images is necessary.

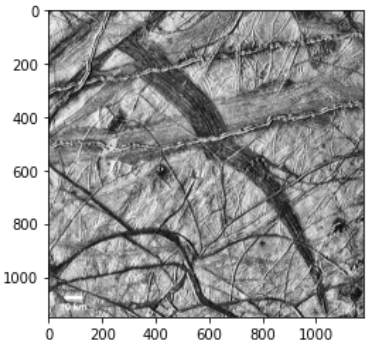
Contrast and brightness Optimization (Histogram Equalization)

Histogram equalization is a technique that stretches the entire spectrum of pixels (0-255) of an image to improve the contrast and appearance (Garg and Jain, 2017).

Adaptive histogram equalization is a modification of traditional histogram equalization. For this technique, the image is divided into multiple small blocks and each block is histogram equalized as usual. Contrast limited adaptive histogram is a variant of adaptive histogram equalization that adds a limitation of contrast to avoid the amplification of noise (Zuiderveld, 1994).



(a) Original image before contrast and brightness optimization



(b) Image After optimization

Fig. 1 Applying contrast limited adaptive histogram equalization technique to optimize contrast and brightness

2.2 Unsupervised-learning:

The goal of unsupervised-learning is to gain useful information about the underlying data structure (Srinidhi, Ciga and Martel, 2019). For this project, we implemented fully unsupervised-learning strategies to extract features from images and then cluster images into groups with these features.

2.2.1 Dimension Reduction

For high-dimensional (number of dimensions more than 10) datasets, we performed dimension reduction before clustering to avoid the effect called *'curse' of dimensionality* (Beyer *et al.*, 1999). When dimensionality goes high enough, even in cases where a well-defined nearest neighbor exists, the difference in distance between the nearest neighbor and any other point in the data set is very small (Beyer *et al.*, 1999).

So the dimension reduction, which transforms data from high-dimensional space into low-dimensional space and keeps the important properties of original data, is essential. Despite principal component analysis (PCA), the most popular linear dimension reduction method, we explored the following models based on neural networks in order to learn latent representations (features) by training.

2.2.2 Restricted Boltzmann Machine (RBM)

Restricted Boltzmann machines (RBM) are two-layer, nonlinear, and unsupervised networks that learn features by maximizing the likelihood of data (Hinton and Salakhutdinov, 2006).

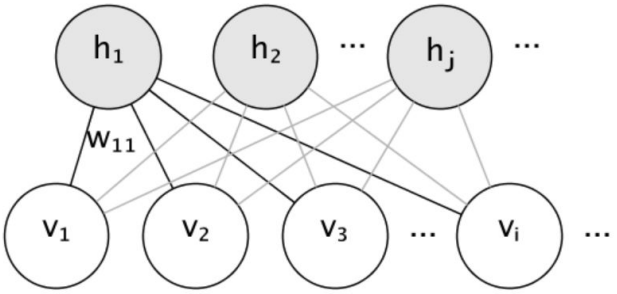


Fig. 2 The graphical model of an RBM (Pedregosa *et al.*, 2011)

The word *restricted* means that there is no direct interaction between hidden units or between visible units.

For our project, the pixels of images correspond to the 'visible' units of the RBM as their states are observed (Hinton and Salakhutdinov, 2006); the features we wanted to extract correspond to the 'hidden' units. The quality of a joint assignment is measured by the energy function (Hinton and Salakhutdinov, 2006); (Pedregosa *et al*., 2011):

$$E(\mathbf{v}, \mathbf{h}) = -\sum_i \sum_j w_{ij} v_i h_j - \sum_i b_i v_i - \sum_j c_j h_j$$

where $\mathbf{v_i}$ and $\mathbf{h_j}$ are the binary states of pixel **i** and feature **j**, $\mathbf{b_i}$ and $\mathbf{b_j}$ are their biases, and $\mathbf{w_{ij}}$ is the weight between them.

And the joint probability of the model is defined in terms of energy (Hinton and Salakhutdinov, 2006); (Pedregosa *et al.*, 2011):

$$P(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z}$$

The parameters are learned using an algorithm called Stochastic Maximum Likelihood (SML) or Persistent Contrastive Divergence (PCD) (Tieleman, 2008).

Stacked RBMs

We can learn a stack of restricted Boltzmann machines, each having only one layer of RBM. The learned feature activations of one RBM are the 'inputs' for training the next RBM in the stack ((Hinton and Salakhutdinov, 2006).

2.2.3 Autoencoder

An autoencoder is a neural network that is trained to copy its input to its output (Goodfellow, Bengio and Courville, 2016). It has a hidden layer that describes the code used to represent the input (Goodfellow, Bengio and Courville, 2016). And the network is composed of two parts: an encoder that encodes the inputs into code, and a decoder that outputs a reconstruction of the original input (Goodfellow, Bengio and Courville, 2016).

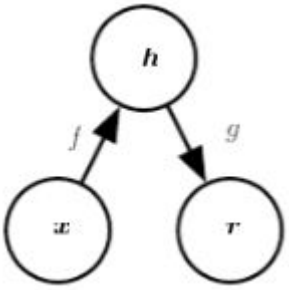


Fig 3. A general structure of an autoencoder (Goodfellow, Bengio and Courville, 2016)

The autoencoder maps an input **x** to an output (called reconstruction) **r** through an internal representation or code **h**. The general structure of the autoencoder has two parts: the encoder ***f*** (mapping **x** to **h**), and the decoder ***g*** (mapping **h** to **r**) (Goodfellow, Bengio and Courville, 2016). **H** is named as *code, latent variables, or latent representation* (Goodfellow, Bengio and Courville, 2016).

Autoencoders have been massively used in dimension reduction or feature learning (Goodfellow, Bengio and Courville, 2016). We can obtain useful features from the autoencoder by constraining **h** to have a smaller dimension than **x**, it forces the autoencoder to learn the most salient features of the training data (Goodfellow, Bengio and Courville, 2016). Autoencoders with nonlinear encoder functions and nonlinear decoder functions learn a more powerful representation of inputs than PCA. However, if the encoder and decoder have too much capacity, *the autoencoder can learn to merely copy inputs without extracting useful information about the distribution of the data* (Goodfellow, Bengio and Courville, 2016).

Denoising autoencoder

A denoising autoencoder is a variant of the basic model of the autoencoder. In the basic model, autoencoders minimize the loss between the reconstructed outputs and original inputs (Goodfellow, Bengio and Courville, 2016):

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))),$$

where L is a loss function calculating how much g(f(x)) for being dissimilar from x, such as the L2 norm of their difference. If the autoencoder has a large capacity, **g** ◦ **f** could be trained to be merely an identity function (Goodfellow, Bengio and Courville, 2016).

A denoising autoencoder (DAE) takes inputs with some form of noise and reconstructs the uncorrupted version of inputs (Goodfellow, Bengio and Courville, 2016). The loss function then becomes (Goodfellow, Bengio and Courville, 2016):

$$L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}}))),$$

Where $\tilde{\boldsymbol{x}}$ is a copy of $\boldsymbol{x}$ that has been corrupted by some form of noise. Denoising autoencoders undo this corruption rather than simply copying their input and forces f and g to implicitly learn the structure of inputs (Goodfellow, Bengio and Courville, 2016).

Sparse autoencoder

A sparse autoencoder is another variant of a basic autoencoder, whose training criterion involves a sparsity penalty $\Omega(\boldsymbol{h})$ on the code layer **h**, in addition to the reconstruction error (Goodfellow, Bengio and Courville, 2016):

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h}),$$

The penalty encourages the model to activate some neurons of the network based on inputs while forcing all other neurons to be inactive (Deng, Fan and Zeng, 2017).

This sparsity of activation can be achieved by formulating the penalty terms in different ways. One way to do it is by exploiting the Kullback-Leibler (KL) divergence. Another way to achieve sparsity in the activation of the hidden unit is by applying L1 or L2 regularization terms on the activation, scaled by a certain parameter λ (Arpit *et al.*, 2015):

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') + \lambda \sum_i |h_i|$$

The model is forced to learn the unique statistical features of the input by this sparsity constraint.

Denoising autoencoders and sparse autoencoders are regularized autoencoders that learn the sparse representation of inputs (Arpit *et al.*, 2015) and they avoid the problem of simply copying inputs to outputs.

2.3 Clustering Algorithm

After extracting the features from the dataset, clustering algorithms are applied to group the learned features.

The clustering algorithm is an unsupervised machine learning strategy that includes two types: hierarchical and non-hierarchical (KETCHEN Jr. and SHOOK, 1996).

2.3.1 Non-hierarchical Method

Non-hierarchical algorithms (also referred to as K-means or iterative methods) seek to divide M points in N dimensions into K clusters and minimize the within-cluster sum of squares (KETCHEN Jr. and SHOOK, 1996). This algorithm generally takes the following steps: 1. Randomly initialize K cluster centers; 2. For each point measures its Euclidean distance from itself to the cluster centers, and finds the closest cluster; 3. Update the cluster centers to be the averages of points contained within them; 4. Repeat from step 2 (Hartigan and Wong, 1979).

2.3.2 Hierarchical Method

The hierarchical method aims to build a hierarchy of clusters (Rokach and Maimon, 2005). There are two types of hierarchical methods: agglomerative and divisive. For the agglomerative strategy, each observation starts in its own cluster and merges with its neighbor as one moves up the hierarchy. The divisive strategy goes in the opposite

direction: all observation starts in one single cluster and is split recursively as one moves down the hierarchy (Rokach and Maimon, 2005).

2.3.3 Determine the number of clusters in a data set

One of the most difficult problems in clustering analysis is determining the number of groups in a data set (Sugar and James, 2003).

Dendrograms

A variety of techniques are available to determine the number of clusters in a data set. When applying hierarchical methods, we can start by visually inspecting a dendrogram, a graph of the order that observations join clusters and the similarity of observations joined (KETCHEN Jr. and SHOOK, 1996).

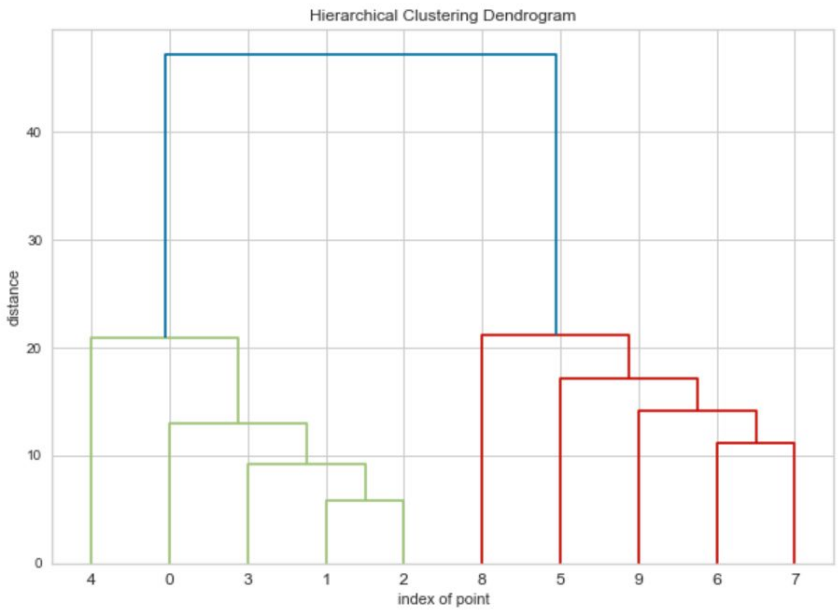


Fig. 4 An example of hierarchical clustering dendrogram

The y-axis of Figure 4 shows the distance between the clusters and the x-axis shows the indexes of observations. A large increase in the distance between the joined clusters suggests that dissimilar clusters have been merged (KETCHEN Jr. and SHOOK, 1996). And by observing the dendrogram, one can set a threshold of the distances between clusters, and the number of clusters can be determined according to the threshold.

Elbow Method

For non-hierarchical clustering, one of the simplistic methods for determining the number of the groups in a data set is measuring the average within-cluster distance of a specific value for k, and plot the distance versus the number of clusters.

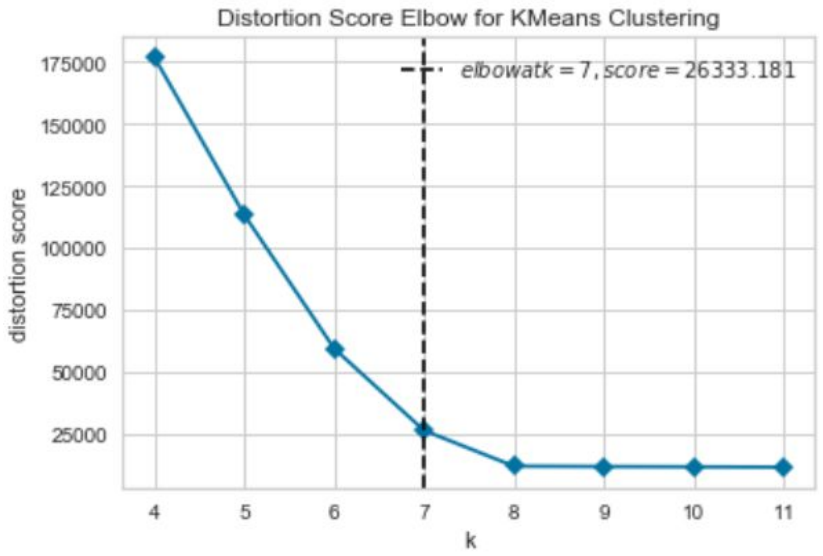


Fig. 5 An example of distortion score elbow for K-means Clustering

Obviously, every increase in the number of groups results in a reduction of average within-cluster distance, but at some point, it would go overfitting. The appropriate number of clusters is found at the 'elbow' of the graph. In Figure 5, the average within-cluster distance decreases rapidly before the 'elbow' and slows down after the 'elbow'.

2.4 Semantic Segmentation

Image segmentation is a process that involves partitioning images into multiple segments or objects (Minaee *et al.*, 2020). Numerous image segmentation techniques have been proposed in the literature, from the earliest methods, such as K-means clustering, histogram-based bundling, to more advanced algorithms such as conditional and Markov random fields (Minaee *et al.*, 2020). However, in recent years, deep learning networks have promoted a new generation of image segmentation with significant performance improvements (Minaee *et al.*, 2020).

Semantic segmentation can be formulated as a classification problem of pixels with semantic labels: image classification predicts a single label for the entire image, semantic segmentation instead performs pixel-level semantic labeling for all image pixels (Minaee *et al.*, 2020).

More than a hundred of deep learning-based semantic segmentation methods have been proposed until 2019, including Fully Convolutional Networks, Convolutional Models with Graphical models, Encoder-Decoder based models, Dilated Convolutional models and DeepLab family, Recurrent Neural Network Based Models, Attention-based models, Generative models, and Adversarial training (Minaee *et al.*, 2020).

transfer learning

In some cases, the deep learning models are trained from scratch on new datasets or applications. However, labeling each training image by pixel is costly and slow. In many cases, there is not enough labeled data available to train a model from scratch and we can apply transfer learning to tackle this problem (Minaee *et al.*, 2020). In transfer learning, a

model previously trained on one problem could be applied to a different problem after a few adjustments. For example, in the image segmentation case, we can use a model trained on ImageNet (a dataset larger than most of image segmentation datasets) as the encoder part of the network, change the output channels to the number of classes of our new dataset, and re-train the model from initial weights (Minaee *et al.*, 2020).

3. Implementation and Software

3.1 Software Development Environment

This software is a standalone code. The machine learning part is developed based on Pytorch which is a Python package that supports Tensor computation (like Numpy) with strong GPU acceleration and provides deep neural networks built on an autograd system.

3.2 Project Workflow and Software Description

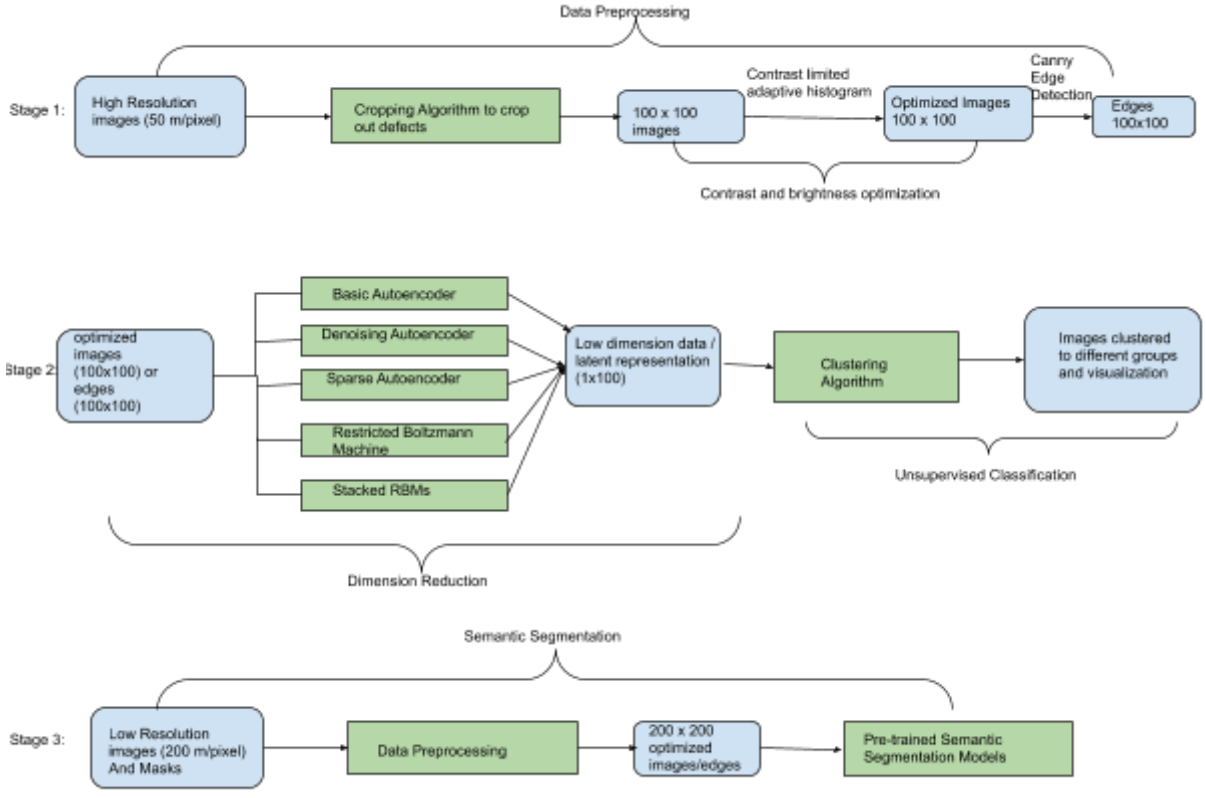


Fig. 6 The workflow of the project

The project workflow is shown above, and the code structure is based on this diagram. The overall software can be regarded as a combination of the following modules (Figure 7): the data pre-processing module, autoencoders & RBMs training module, unsupervised classification & visualization module, semantic-segmentation models training & visualization module. The data pre-processing module prepares the training set for autoencoders & RBMs

training module and semantic-segmentation models training & visualization module. The autoencoder & RBMs training module produces the compressed, low-dimensional latent representation of inputs and then the unsupervised classification & visualization module applies clustering algorithms with the latent presentations to classify the images. The semantic-segmentation model training and visualization module saves the models with weights after training and makes predictions using the saved model as well as visualizing the predictions.

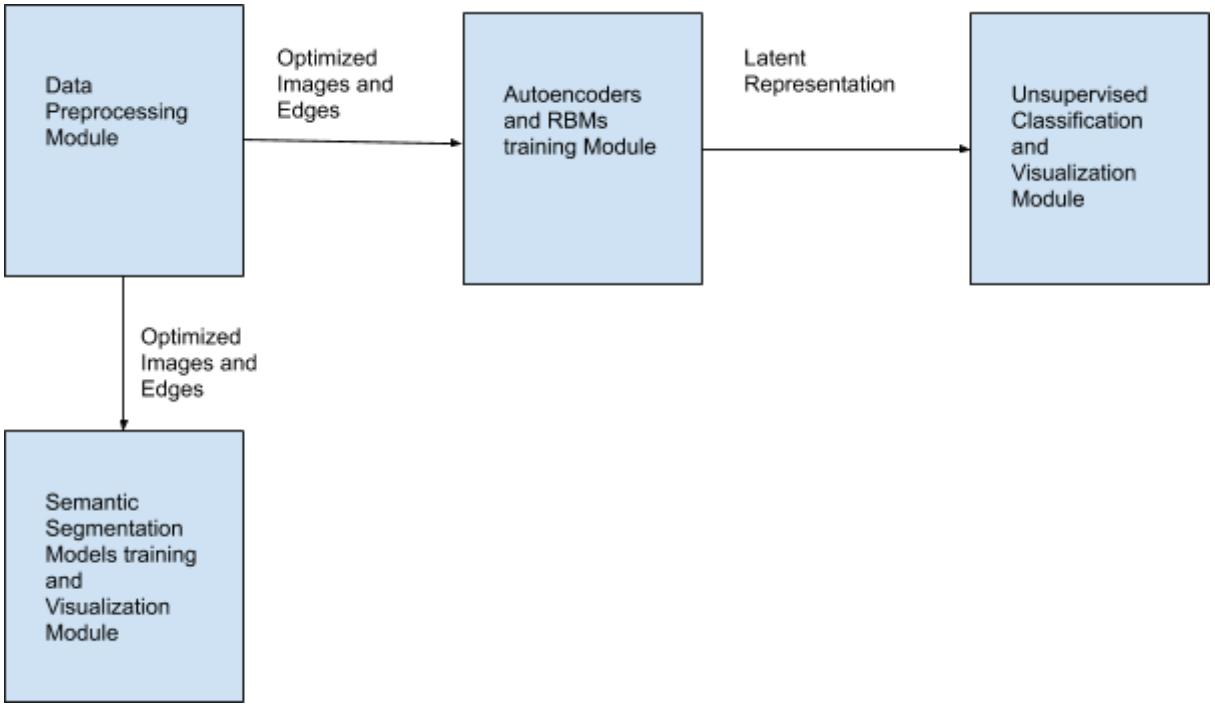


Fig. 7 The general structure of the software

3.3 Data Preprocessing

3.3.1 Recursive cropping algorithms to find the largest useful rectangle

Most images provided for training have large black/white areas and these images are regarded as images with defects. To deal with this problem, horizontal-cut recursion and vertical-cut recursion algorithms are developed to find the largest useful rectangle from those images with defects (Figure 8).

3.3.2 Contrast and brightness optimization and edge detection

After cropping out the defects from images, we need to address the non-uniform illumination problem. We use the built-in CLAHE function from the OpenCV library to optimize the contrast and brightness of images and apply the canny edge detection technique (R *et al.*, 2019) to extract edges from the optimized images. **Then we got optimized images and edges as our two types of training data for our models.**

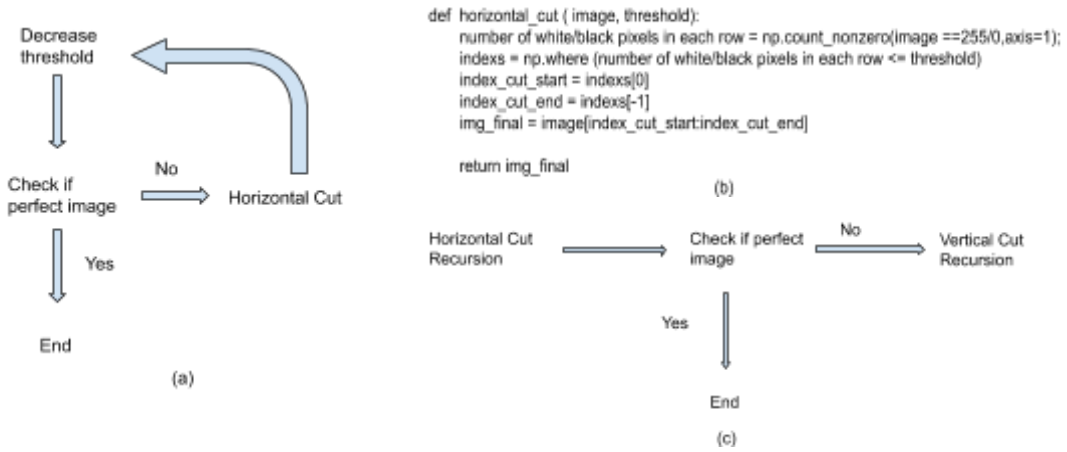


Fig. 8 The algorithms of recursive cutting. (a) shows the structure of the horizontal recursive cutting algorithm, the vertical recursive cutting follows the same structure; (b) the pseudo-code of one horizontal cut and the vertical cut uses a similar structure; (c) images are first to be cut horizontally and then cut vertically.

3.3.3 Training Dataset

For unsupervised classification tasks, we use images with 50m/pixel resolution as our training dataset. To ensure each image contains only one possible type of fracture, the training images for classification are cut to 100 pixels wide and 100 pixels high, a size small enough to limit the number of fractures in one image.

As for image segmentation tasks, we use images with 200m/pixel resolution as our training dataset., which means training images for segmentation have a much larger scale than that of images for classification and we are inspecting the fractures on Europa from a higher angle. Different from unsupervised classification tasks, the image segmentation tasks are fully-supervised and labels are needed. We use the open-source, Computer Vision Annotation Tool (CVAT), to label the pixels of our images. The labels are based on the classification of fractures from the paper *'Analysis of very-high-resolution Galileo images and implications for resurfacing mechanisms on Europa' (Leonard, Pappalardo and Yin, 2018; see figure 9)*, and we conclude those classifications of fractures into 5 classes: plain (label '0'), band (label '1'), ridge (label '2'), chaos (label '3') and crater (label '4').

3.4 Autoencoder

The architecture of the autoencoder was designed from scratch. It is composed of three convolutional layers, two linear layers, and three transposed convolutional layers. During the downsampling process (Figure 10), the number of output channels is increasing and the sizes of images in x,y dimension are decreasing. Two linear layers are put in the middle of the model so that we could easily control the size of 'code' or the dimension of the latent representation of inputs. Batch normalization technique is applied after each convolutional layer to mitigate the internal covariate shift problem by fixing the means and variances of each layer's inputs (Ioffe and Szegedy, 2015). Apart from the last convolutional layers, most of the layers were activated by the PReLU function. Unlike the most known activation

function, ReLU, PReLU doesn't make the negative inputs dead and the slope can be learned during training (He *et al.*, 2015). Dropout technique is also applied to reduce overfitting by randomly dropping units (along with their connections) from the neural network during training (Srivastava *et al.*, 2014).

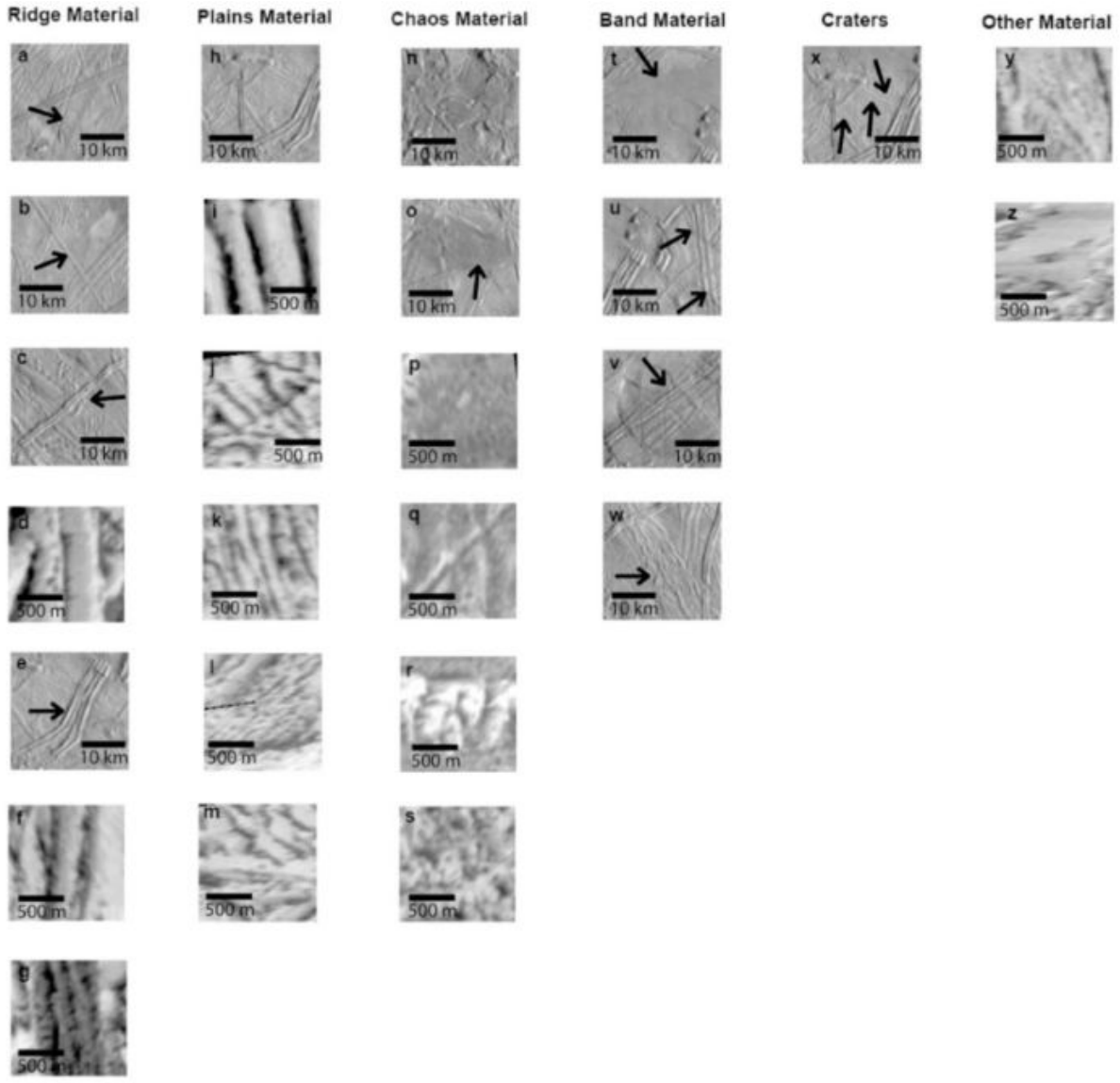


Fig. 9 Classification of material units based on geomorphology, texture, albedo, and size (Leonard, Pappalardo and Yin, 2018); our labels for semantic segmentation are based on these.

Sparse Autoencoder and Denoising Autoencoder

Both sparse autoencoder and denoising autoencoder use the same architecture as our basic autoencoder.

The denoising autoencoder takes noisy images as input, gets the hidden latent representation, and then reconstructs the original images. For our project, we use NOISE_FACTOR to control the amount of noise we add to the images. And to ensure the value of the inputs do not go beyond the range [0,1], we clip the images using np.clip() function.

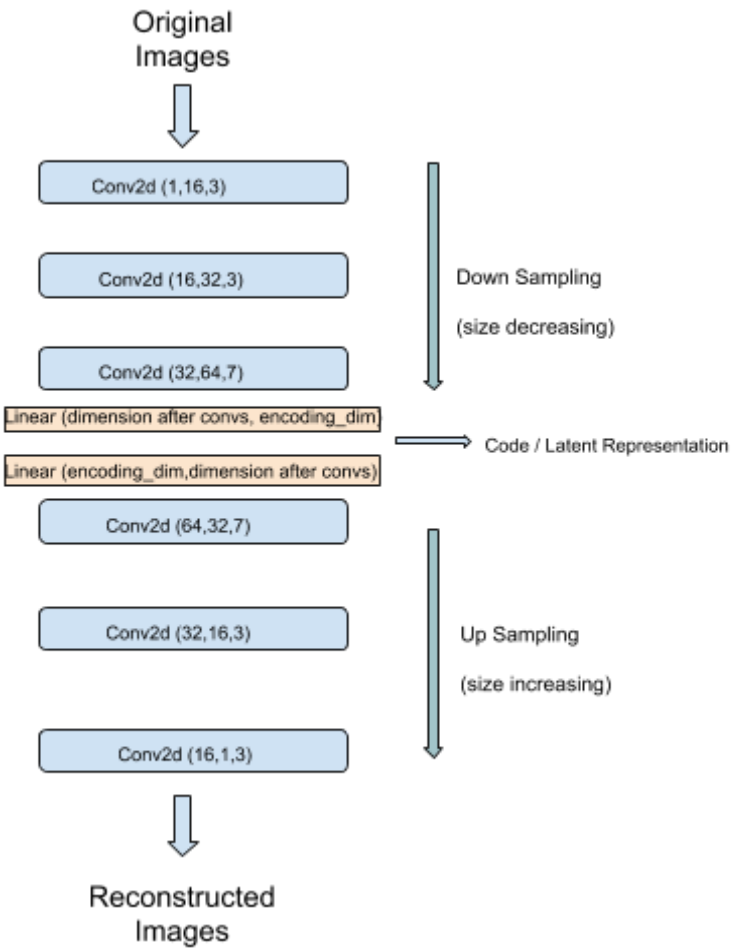


Fig.10 The structure of the basic autoencoder. It consists of both convolutional layers and linear layers.

Sparse autoencoder was implemented by adding the L1 sparse penalty to the activations of the neuron after the PReLU function. The final loss then became:

$$Final\_loss = MSELoss + \lambda * \sum |wi|$$

It should be noted that the software can combine the sparse and denoising functionality together to form a sparse-denoising autoencoder. One can simply set the parameter *add_sparsity* to True and *denoising* to True (Figure 11.a).

Pseudo code of training functions of autoencoders:

```
def train(model,optimizer,criterion, data_loader,add_sparsity=False, denoising=False,lambda=0,NOISE_FACTOR=0.5):
    For data in data_loader:

        Image,_ = data

        If denoising = True:
            Image_noisy = Image + Gaussian_noise*NOISE_FACTOR
            Outputs = Model (Image_noisy)
            L1_loss = Model.sparse_loss (Image_noisy)
        Else:
            Outputs = Model(Image)
            L1_loss = Model.sparse_loss(Image)

        MSE_loss = criterion (Outputs, Image)

        If add_sparity = True:
            Loss = lambda * L1_loss + MSE_loss
        Else:
            Loss = MSE_loss

        optimizer.zero_grad()

        loss.backward()

        optimizer.step()
```

(a)

```
def get_output_shape(model, image_dim):
  return model(torch.rand(*(image_dim))).data.shape


class autoencoder(nn.Module):
  """
  this class is able to generate an autoencoder model that compress the image to a specified dimension-number (encoding_dim)

  """
  def __init__(self,input_dim,encoding_dim):  # encoding_dim: the size after encoding , size : the size before encoding
    super(autoencoder,self).__init__()
    self.enc1 = nn.Sequential(nn.Conv2d(1, 16, 3, stride=2, padding=1),nn.PReLU(),nn.GroupNorm(16,16),nn.PReLU(),nn.Dropout(p=0.2))
    self.enc2 = nn.Sequential(nn.Conv2d(16, 32, 3, stride=2, padding=1),nn.PReLU(),nn.GroupNorm(32,32),nn.PReLU(),nn.Dropout(p=0.2))
    self.enc3 = nn.Sequential(nn.Conv2d(32, 64, 7),nn.PReLU(),nn.GroupNorm(64,64),nn.PReLU())

    self.num_shape_after_cnn1 = get_output_shape(self.enc1,input_dim)
    self.num_shape_after_cnn2 = get_output_shape(self.enc2,self.num_shape_after_cnn1)
    self.num_shape_after_cnn3 = get_output_shape(self.enc3,self.num_shape_after_cnn2)

    self.num_shape_after_cnn_final = self.num_shape_after_cnn3

    self.num_features_before_fcnn = np.prod(list(self.num_shape_after_cnn_final)) # flatten
    print(self.num_features_before_fcnn)
    self.fc1 = nn.Sequential(nn.Linear(in_features=self.num_features_before_fcnn, out_features=encoding_dim),nn.PReLU(),nn.Dropout(p=0.2))
    self.fc2 = nn.Sequential(nn.Linear(in_features=encoding_dim, out_features=self.num_features_before_fcnn),nn.PReLU(),nn.Dropout(p=0.2))

    self.dec1 = nn.Sequential(nn.ConvTranspose2d(64, 32, 7),nn.PReLU(),nn.GroupNorm(32,32),nn.PReLU())

    self.dec2 = nn.Sequential(nn.ConvTranspose2d(32, 16, 3, stride=2, padding=1,
output_padding=1),nn.PReLU(),nn.GroupNorm(16,16),nn.PReLU(),nn.Dropout(p=0.2))

    self.dec3 = nn.Sequential(nn.ConvTranspose2d(16, 1, 3, stride=2, padding=1, output_padding=1),nn.Sigmoid())
```

(b)

Fig. 11 (a) The pseudo-code of training functions of autoencoders, we can easily build a basic autoencoder, denoising, or sparse autoencoder by controlling the parameter of the training function. (b) A code snippet of the autoencoder class: the structure of the autoencoder can be changed according to the dimension of latent representation we would like to extract.

3.5 RBM and Stacked RBMs

The RBM was built by specifying the number of visible units to the size of our images and the number of hidden units to the size of the latent representation we want.

The Stacked RBMs was composed of four layers of RBM. The learned features of each layer are the inputs or visible units for the next RBM layer. The number of learned features of each layer is decreasing as inputs propagate the network: the first layer extracts 5000 features, the second layer outputs 2000 features, the third layer produces 1000 features, and at the last layer we get 100 features (Figure 12).

3.6 Image Segmentation and Transfer Learning

Four segmentation models have been trained for image segmentation tasks including DeepLabV3-Resnet101, DeepLabV3-Resnet50, FCN-Resnet101, and FCN-Resnet50. The models have been pre-trained on a subset of COCO train 2017 on the 20 categories that are present in the Pascal VOC dataset. To fit our Europa satellite image dataset, we modified the number of output channels of the classifier to the fracture types on Europa (Figure 13).

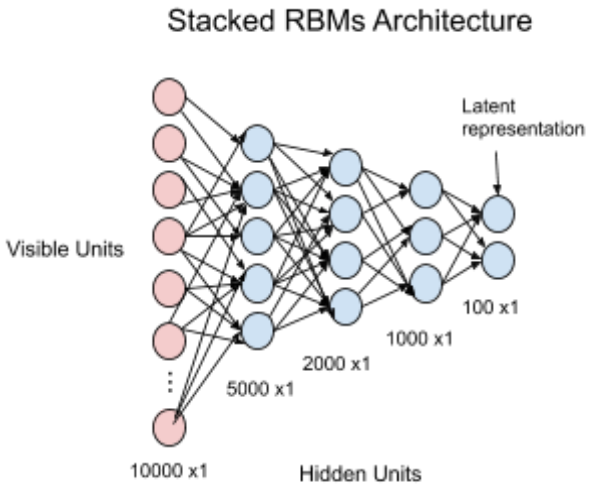


Fig. 12 The architecture of Stacked RBMs, which consists of four layers of RBM. The images are flattened before training.

```
def create_segmentation_model(network_type='DeepLabV3',backbone='Resnet101',outputchannels=5):
    if network_type == 'DeepLabV3':
        if backbone == 'Resnet101':
            print('Creating DeepLabV3-resnet101...')
            model = models.segmentation.deeplabv3_resnet101(pretrained=True, progress=True)
        if backbone == 'Resnet50':
            print('Creating DeepLabV3-resnet50...')
            model = models.segmentation.deeplabv3_resnet50(pretrained=True, progress=True)
        model.classifier = DeepLabHead(2048, outputchannels)
    if network_type == 'FCN':
        if backbone == 'Resnet101':
            print('Creating FCN-resnet101...')
            model = models.segmentation.fcn_resnet101(pretrained=True, progress=True)
        if backbone == 'Resnet50':
            print('Creating FCN-resnet50...')
            model = models.segmentation.fcn_resnet50(pretrained=True, progress=True)
        model.classifier = model.classifier = FCNHead(2048, outputchannels)

    return model
```

Fig. 13 A code snippet of the creating segmentation models function. There are four options for pre-trained semantic segmentation models.

3.7 Novelty and Creativity of the Software

The novel contributions of this work are highlighted here.

1. Develop recursive cropping algorithms to crop out the largest useful rectangle from images with defects.
2. Build convolutional autoencoders from scratch and the autoencoder can easily switch to denoising or sparse autoencoders.
3. Build stacked RBMs.
4. Extract features from the inputs by various models. And the number of features produced by autoencoders can be specified (Figure 11 b).
5. Functions to create transfer learning-based semantic segmentation models. Models can be easily modified according to the number of classes on a new dataset.

4. Code metadata

Platform

The software is mainly developed using language Python3. In addition, the libraries below are included. As machine learning is heavily involved in our project, the major part of the software is developed under the Google Colab platform. Google Colab platform provides an NVIDIA Tesla P100-PCIE-16GB GPU and users can use up to 12 hours for free per day.

Libraries

The following are utilized in the software:

1. OpenCV 4.1.2
   OpenCV is an open-source library that contains powerful tools to solve computer vision problems.

2. Scikit learn 0.22.2
   Scikit learn is a machine learning library with various classification, regression, and clustering algorithms.

3. Pytorch 1.6.0
   Pytorch is also a machine learning library and has high performance in deep neural networks.

4. PIL 7.0.0
   Python Imaging Library (PIL) provides support for manipulating and saving many different image formats.

5. PyCM 2.8
   PyCM is a multi-class confusion matrix library in Python.

6. livelossplot 0.5.3
   Livelossplot, as the name suggests, is a live-plotting tool to observe the training loss.

7. Numpy 1.18.5

Numpy adds capabilities of operating high-level mathematical operations on multi-dimensional arrays and matrices.

Link to the codes
https://github.com/acse-2019/irp-acse-ll2819

5. Results and Analysis

5.1 Unsupervised Learning and Classification

Image classification without dimension reduction

To prove the importance of dimension reduction before clustering, K-means clustering is applied directly to original images. Distortion score is a metric that computes the sum of squared distances from each point to its assigned center. Intuitively, when the number of clusters (K) increases, the distortion score should decrease. However, as the original images are very high-dimensional (100 pixels wide x 100 pixels high = 10e4 pixels in total) data, there are fluctuations in Figure 14. This phenomenon reflects the 'curse' of dimensionality - 'even though there is a well-defined nearest neighbor, the difference in distance between the nearest neighbor and any other point in the data set is very small' (Beyer *et al.*, 1999). In addition, the calculation precision of computers is limited, when the data is very high dimensional, it may not be able to find the real centers of clusters.

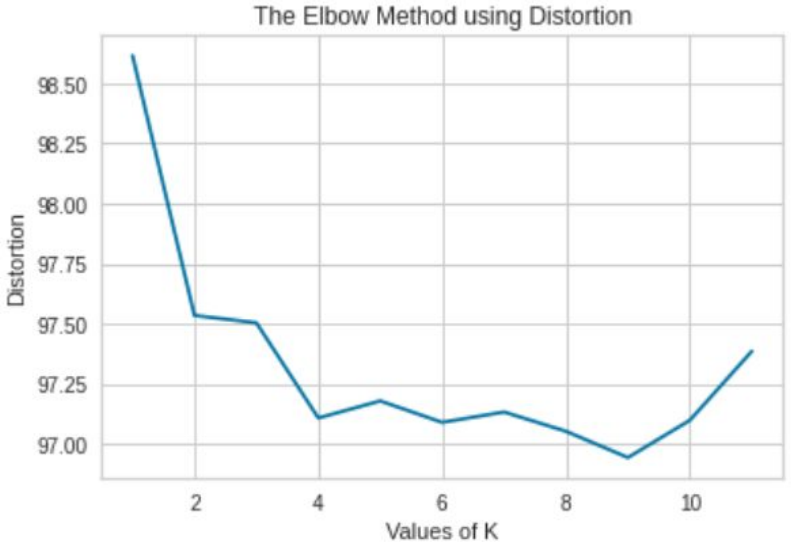
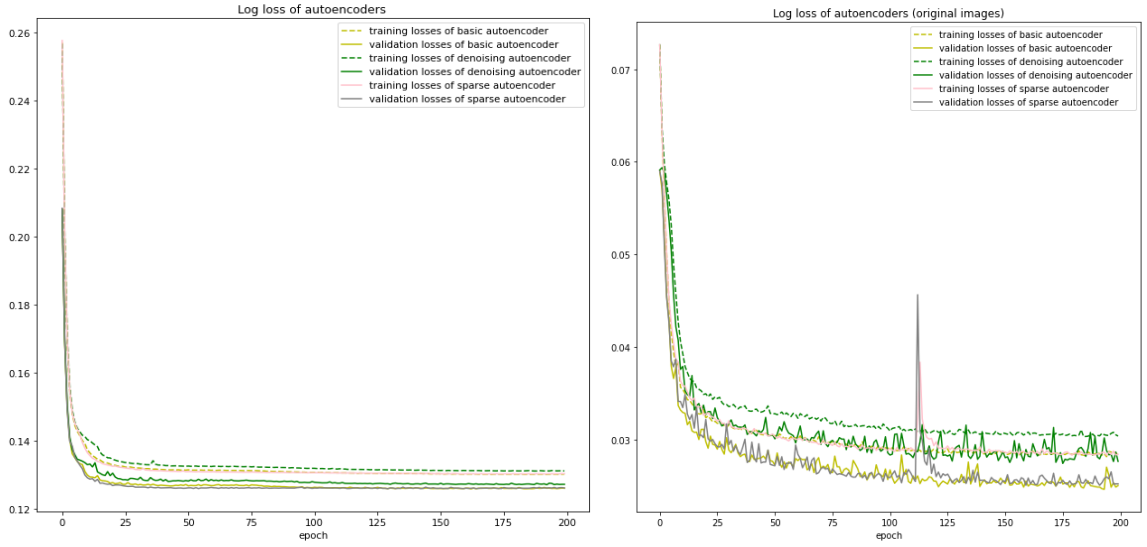


Fig. 14 The distortion score vs the number of clusters in the original images dataset before dimension reduction. The fluctuations of curves suggest the 'curse' of dimensionality.

Thus, implementing dimension reduction techniques is necessary. **We decide to extract 100-dimensional latent representations of the images and all autoencoders and RBMs are trained to achieve this goal.**

5.1.1 Training Autoencoders

Two types of training images are used for training autoencoders, including original images after optimization and edges extracted from the original images. The basic autoencoder, denoising autoencoder, and sparse autoencoder have less training loss and validation loss when trained by original images than trained by edges (Table 1, Table 2). This is probably due to the fact that edges are *binary inputs* (the value of pixels is either 0 or 255) but the original images have more continuous values. Figure 15 below indicates that the training of all the autoencoders converged, and validation losses for all models are smaller than training losses, a success of avoiding overfitting problems.



(a) (b)

Fig. 15 The plots of Losses of training sets and validation sets for autoencoders. (a) The losses when trained with edges; (b) The losses when trained with original images. Overall, the losses when trained with original images (b) are significantly smaller than trained with edges (a).

5.1.2 Training RBM and Stacked RBMs

Since RBM and Stacked RBMs only accept binary inputs, the training of RBM and Stacked RBMs with original images does not converge; thus, we only train RBMs with edges detected from images. One way to test the performance of RBM and stacked RBMs is by passing the training dataset to the models and measuring the likelihood (Pedregosa *et al.*, 2011). After training for 100 epochs, stacked RBMs obtain a much larger average pseudo-likelihood than RBM (Table 2). Thus stacking RBMs have an improved performance than only one layer of RBM.

	Minimum Training Loss	Minimum Validation Loss
Basic Autoencoder	0.028	0.025
Denoising Autoencoder	0.030	0.027
Sparse Autoencoder	0.028	0.025

Table 1. Autoencoders trained with 6691 original images as the training set and 743 original images as the validation set. Their training losses and validation losses are remarkably close.

	Minimum Training Loss	Minimum Validation Loss	Final Mean Pseudo-Likelihood
Basic Autoencoder	0.130	0.126	
Denoising Autoencoder	0.131	0.127	
Sparse Autoencoder	0.130	0.126	
RBM			-4424.76
Stacked RBM			-254.40

Table 2. Autoencoders and RBMs trained with 6691 images of edges as the training set and 743 images of edges as the validation set. All validation losses are smaller than the training losses: overfitting problems are avoided. The final average pseudo-likelihood of the Stacked RBMs is significantly larger than the RBM, which indicates a huge improvement when we have a deeper RBMs model.

5.1.3 Unsupervised Classification Analysis and Comparison of Models

After extracting features from the images by different models, we are ready to visualize and cluster those features as well as classify the corresponding images. Dendrograms are used for observing how well the data can be clustered. When drawing the dendrograms, data are clustered by minimizing the total within-cluster variance (Ward's minimum variance criterion). We also plot the distortion score versus the number of clusters by K-means clustering and see if there is an 'elbow'; if the curve is smooth, it suggests that the data may not consist of any groups.

And to compare the performance of all models, we specifically inspect the case when encoded data are clustered into two groups (Figure 16, Figure 17).

Encoded data from models that are trained with edges

Figure 16 and Figure 17 suggest that if encoded data from edges are clustered into two groups either by K-means or Hierarchical agglomerative clustering (HAC), the distances between the groups have the largest increase. The dendrograms in figure 17 show that

when clustered into two groups, *encoded data produced by stacked RBMs have the smallest ratio of average within-cluster distance to distance between clusters than other models.*

If we cluster the data preliminary using different clustering methods (Figure 18 and Figure 19), for most models, when data are clustered by K-means, clear boundaries between different groups can be observed but groups tend to overlap when clustered by HAC. However, no matter clustered by K-means algorithms or hierarchical clustering, there are clear boundaries between different groups when features are extracted by stacked RBMs. And among all the models, stacked RBMs have densest clusters.

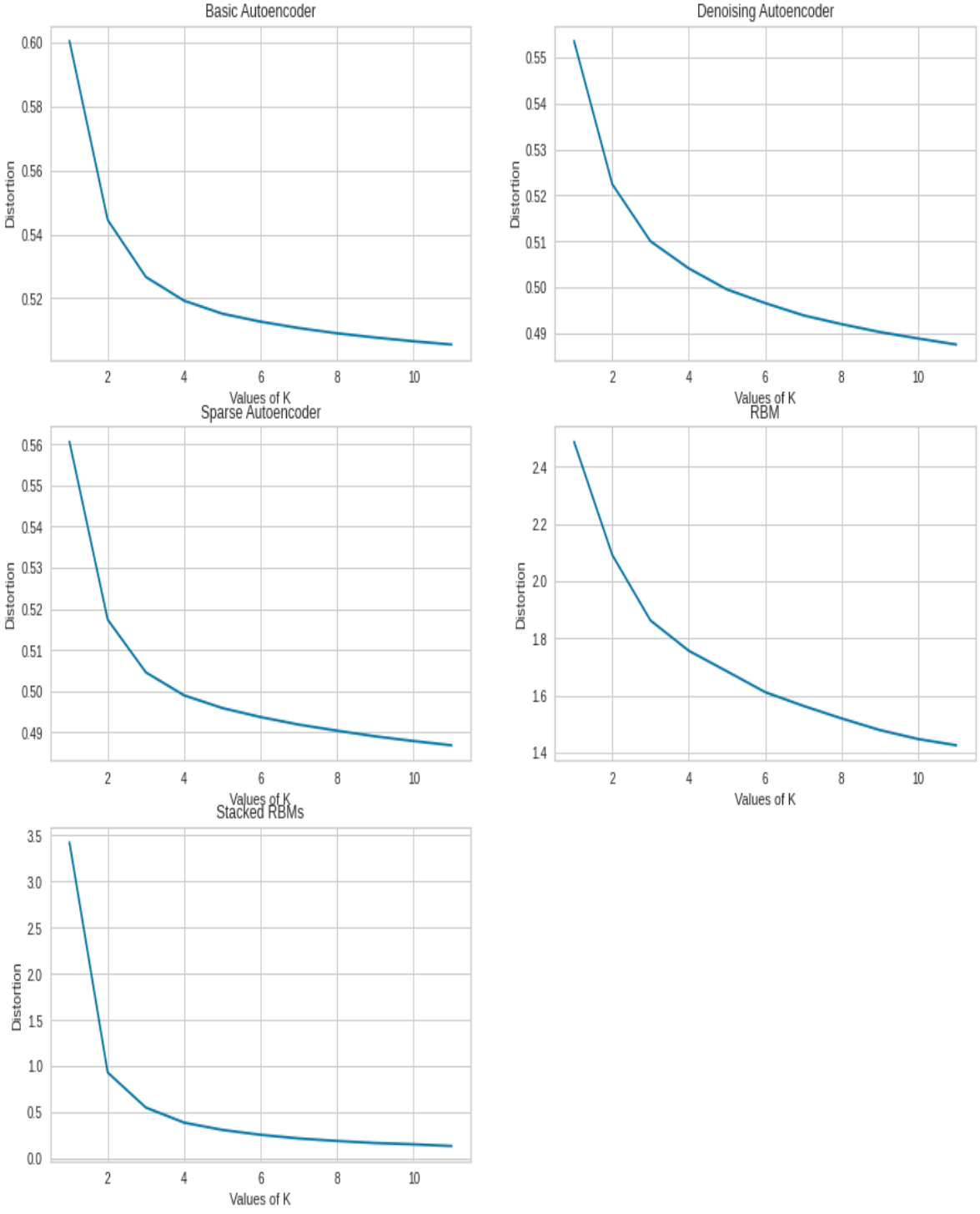


Fig. 16 The Distortion scores vs the number of clusters. All subplots suggest when K = 2, there is a large decrease in the average within-cluster distance. (Features are extracted from edges). Among all models, the decrease of the average within-cluster distance of stacked RBMs is the largest: ⅔ of the diameter of the entire dataset.

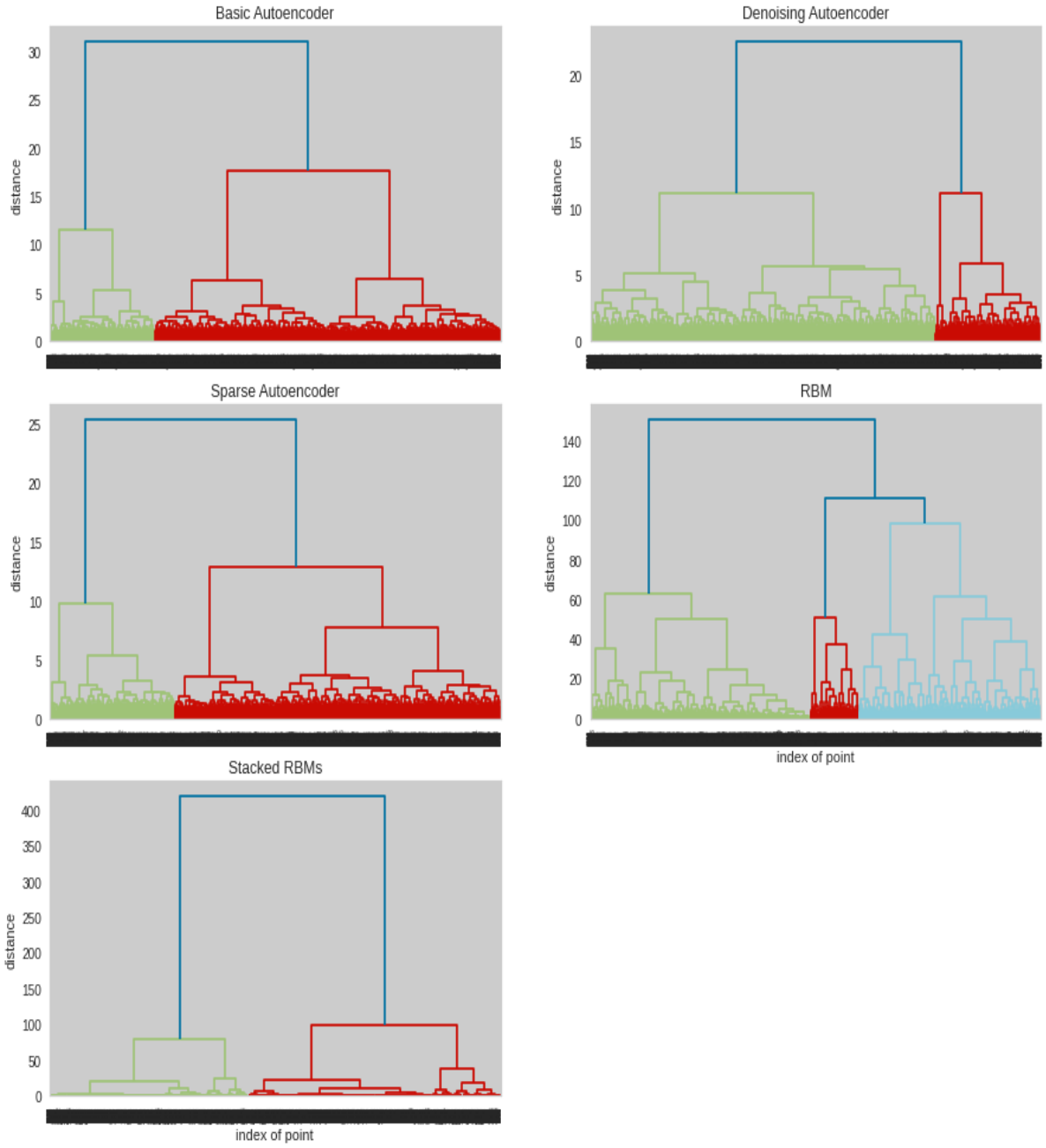


Fig. 17 The dendrograms of encoded data from all sources. Subtitles of each subplot are the 'source' of encoded data: the model where the encoded data are extracted. (Features are extracted from edges). Among all models, stacked RBMs have the largest distance between clusters when data are divided into two groups.

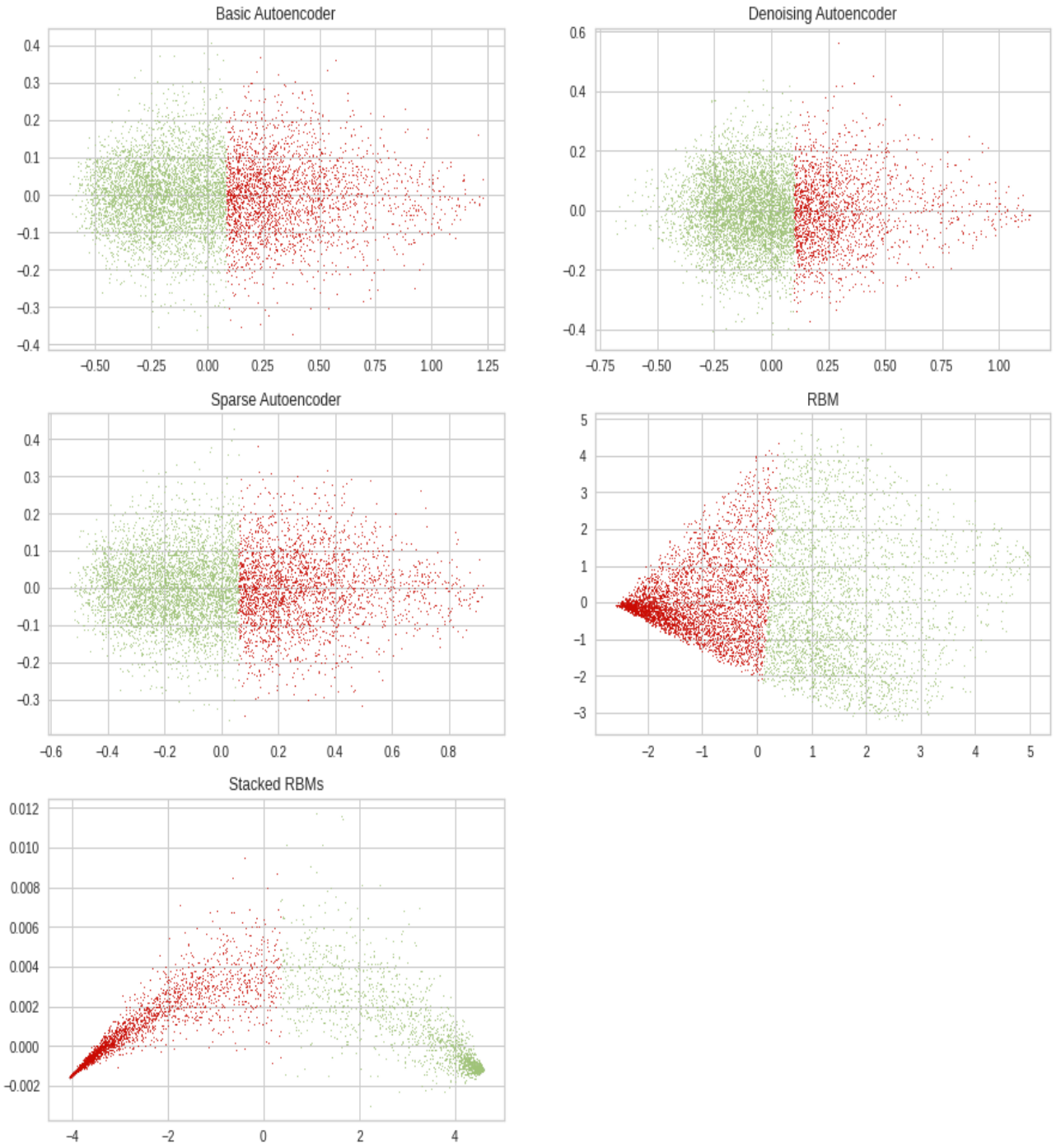


Fig. 18 Visualization of clusters. Encoded data from stacked RBMs has densest clusters. Data are projected to two-dimensional space by PCA, and encoded data are grouped using the K-means clustering algorithm. Subtitles of each subplot are the 'source' of encoded data: the model where the encoded data are extracted. (Features are extracted from edges)

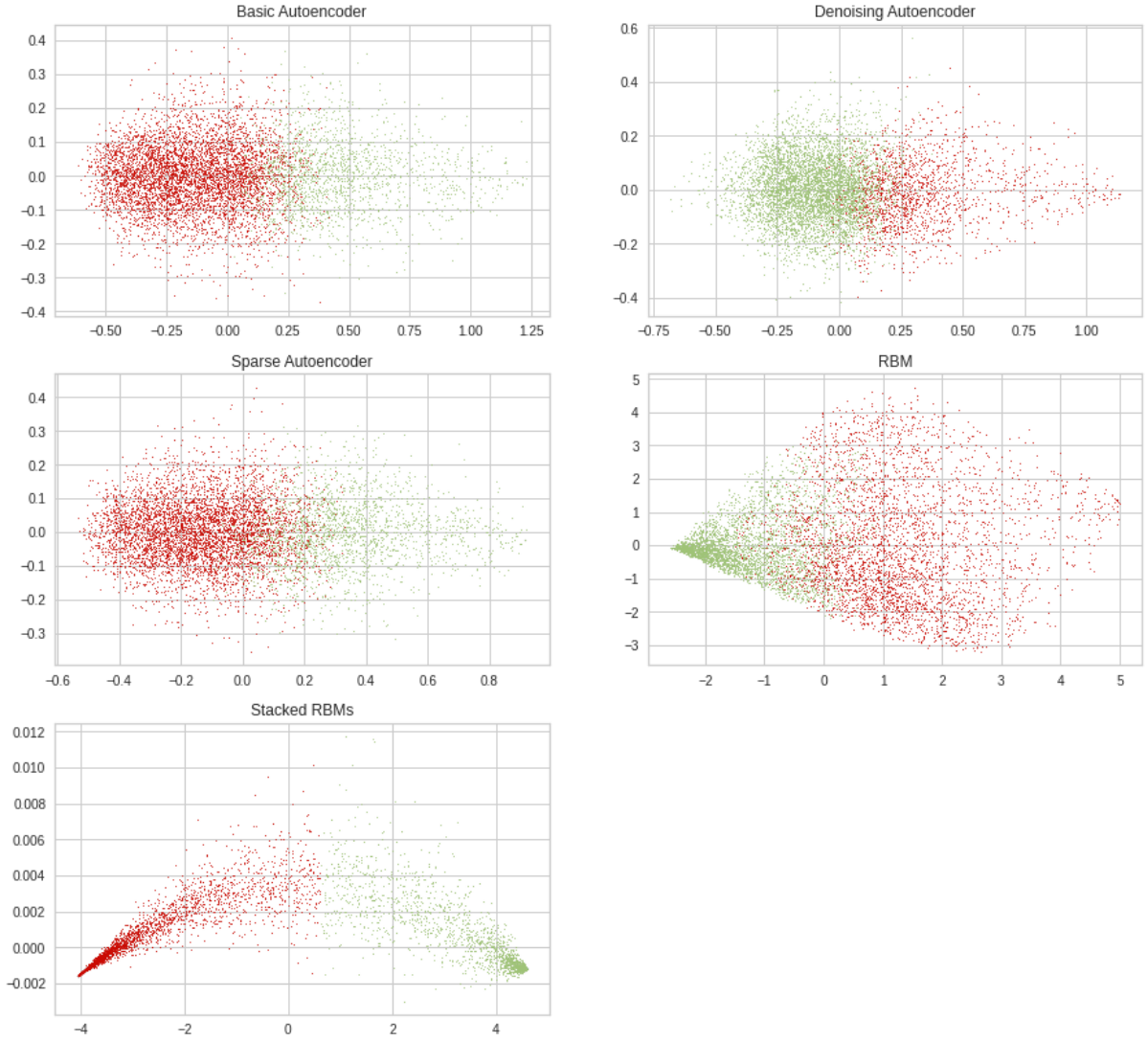


Fig. 19 Visualization of clusters. Only encoded data from stacked RBMs have a clear boundary between clusters. Data are projected to two-dimensional space by PCA, and encoded data are grouped using hierarchical clustering algorithms. Subtitles of each subplot are the 'source' of encoded data: the model where the encoded data are extracted. (Features are extracted from edges)

Encoded data from models that are trained with original images

Since RBM and stacked RBMs only accept binary inputs, only encoded data extracted by autoencoders are analyzed. Even though autoencoders trained by original images have much smaller training loss and validation loss than trained by edges (Table 1, Table 2), when clustered into two groups, the ratio of the average within-cluster distance to the distance between clusters are still larger than stacked RBMs (Figure 17, Figure 20).

But it should be noticed that when trained with relatively continuous data, the denoising autoencoder and the sparse autoencoder tend to extract encoded data that are more clustered (Figure 22). This, to some extent, proves that regularized autoencoders, such as denoising autoencoders and sparse autoencoders, encourage the sparsity of representation (Arpit *et al.*, 2015).

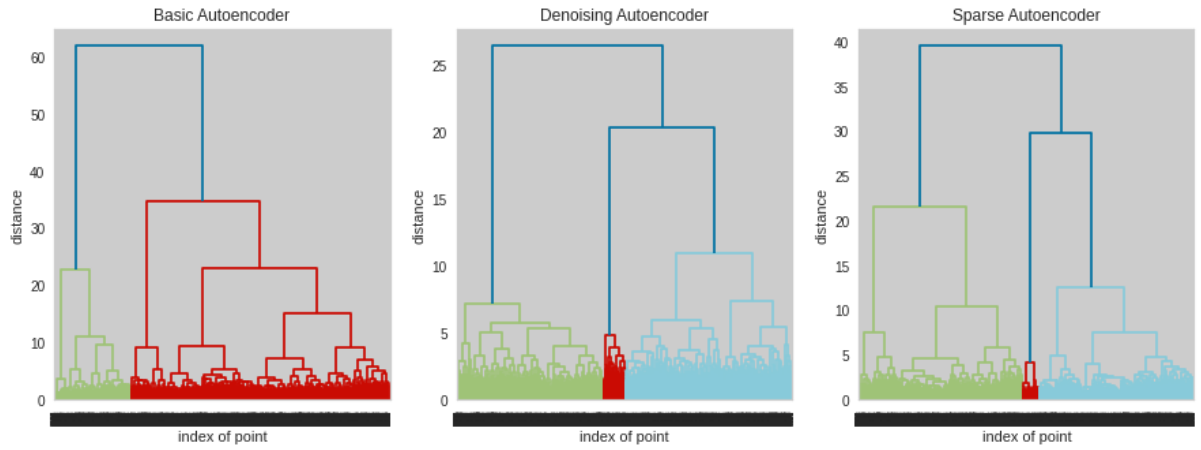


Fig. 20 The dendrograms of encoded data that are extracted by basic autoencoder, denoising autoencoder, and sparse autoencoder. (Features extracted from original images). Clusters of data are not as clear as clusters of data from stacked RBMs.

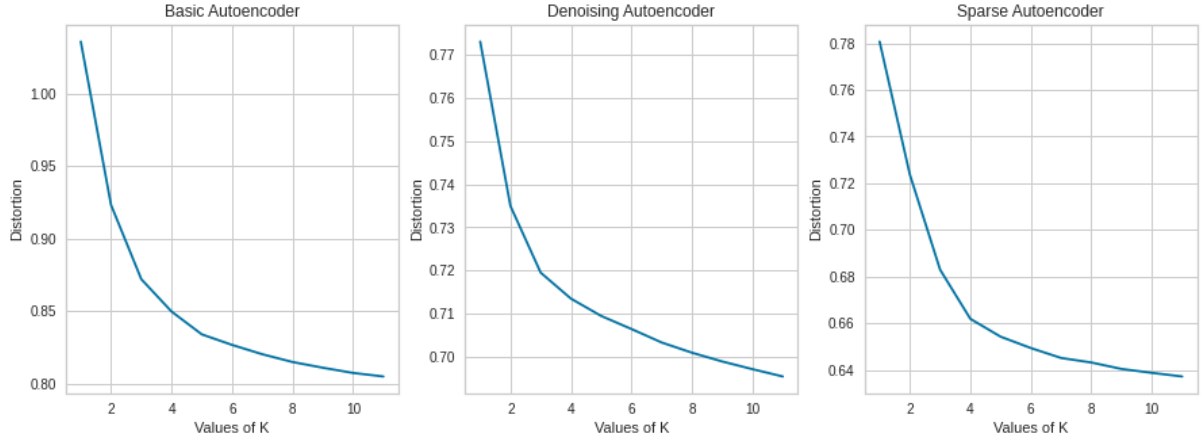


Fig. 21 The Distortion scores vs the number of clusters. (Features extracted from original images)

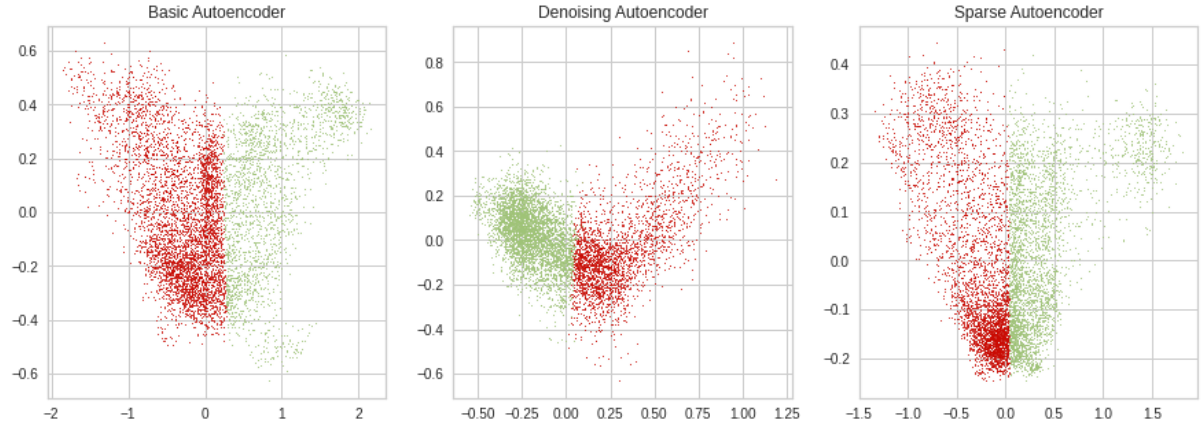


Fig. 22 Visualization of clusters. Data are projected to two-dimensional space by PCA, and encoded data are grouped using K-means clustering algorithms. Subtitles of each subplot are the 'source' of encoded data: the model where the encoded data are extracted. (Features are extracted from original images)

5.1.4 Image classification using encoded data extracted from Stacked RBMs (trained by edges)

From the above analysis, the encoded data extracted from stacked RBMs are easier to cluster and the high pseudo-likelihood of stacked RBMs demonstrates the code is highly representative (Table 2). So the best option among all the models to extract features and classify the Europa images is stacked RBMs.

K-means clustering has the disadvantage that the centroid of clusters is chosen randomly at the start, so we implemented hierarchical agglomerative clustering to group encoded data. As the dendrogram, Figure 23, suggests, we can first cluster the encoded data into two 'large' groups and then observe each individual 'large' group. To simplify, we call the first large group Group A, and the second large group Group B. We have 7434 images in total: 4163 images are clustered into Group A and 3271 images are clustered into Group B.

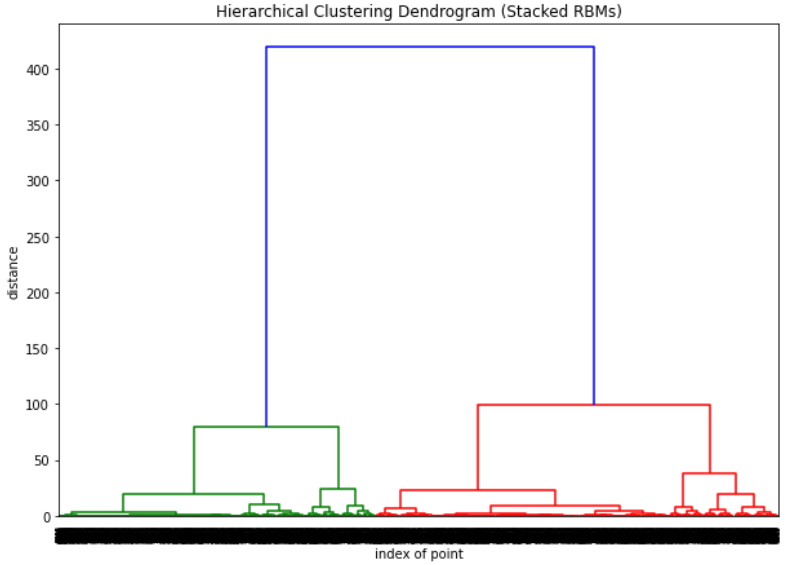


Fig. 23 The dendrogram of encoded data extracted from stacked RBMs. When data is clustered into two or four groups, the distance between clusters has a very large increase.

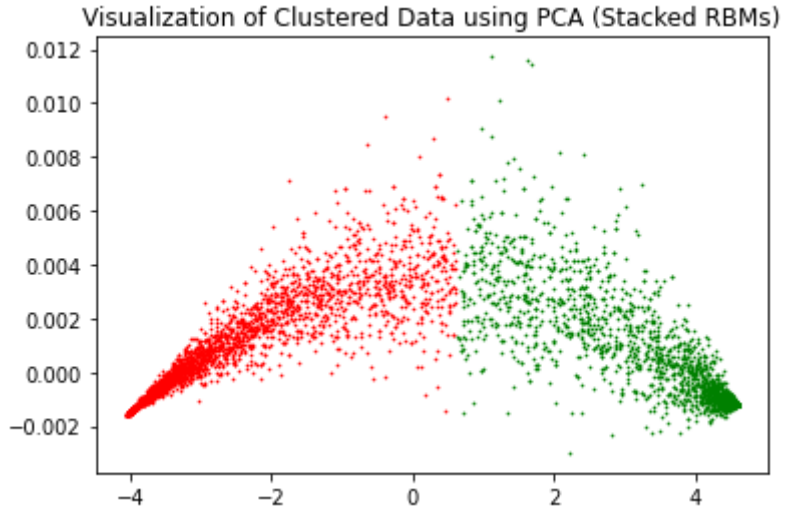


Fig. 24 Visualization of clusters. Data are projected to two-dimensional space by PCA, and encoded data are grouped using hierarchical clustering algorithms.

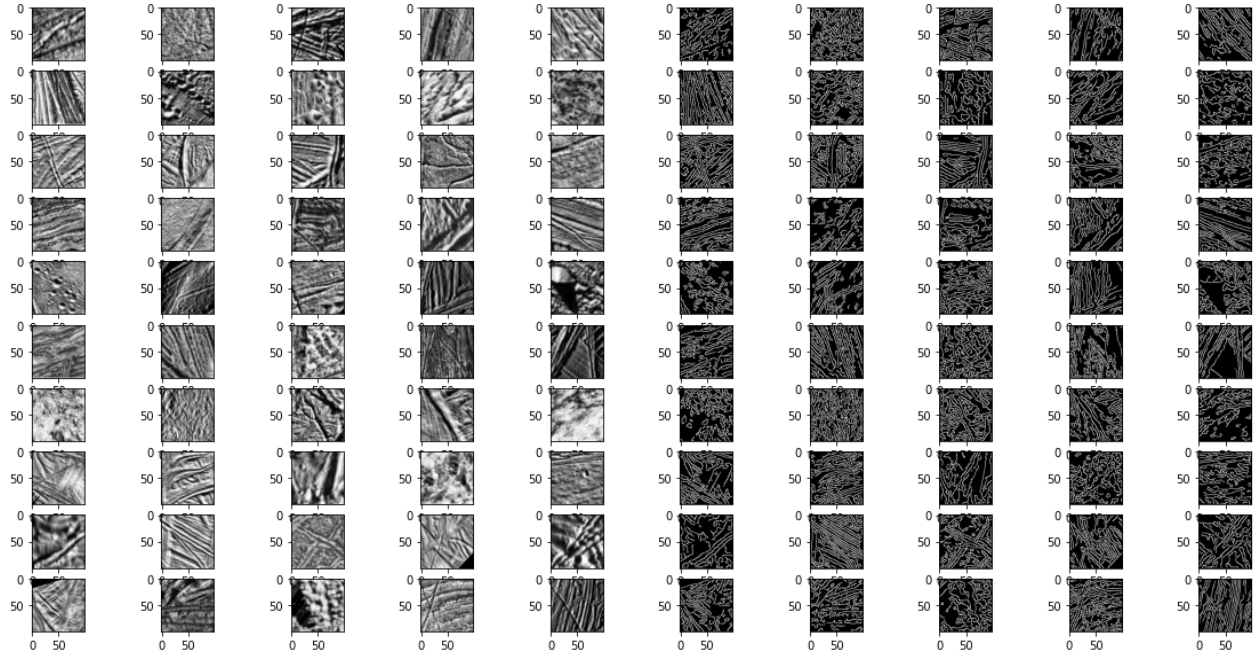


Fig. 25 Some samples of the images that are clustered into the first 'large' group (Group A). The corresponding edges detected from original images are on the right. (See Appendix A for a larger version of the figure and more samples)

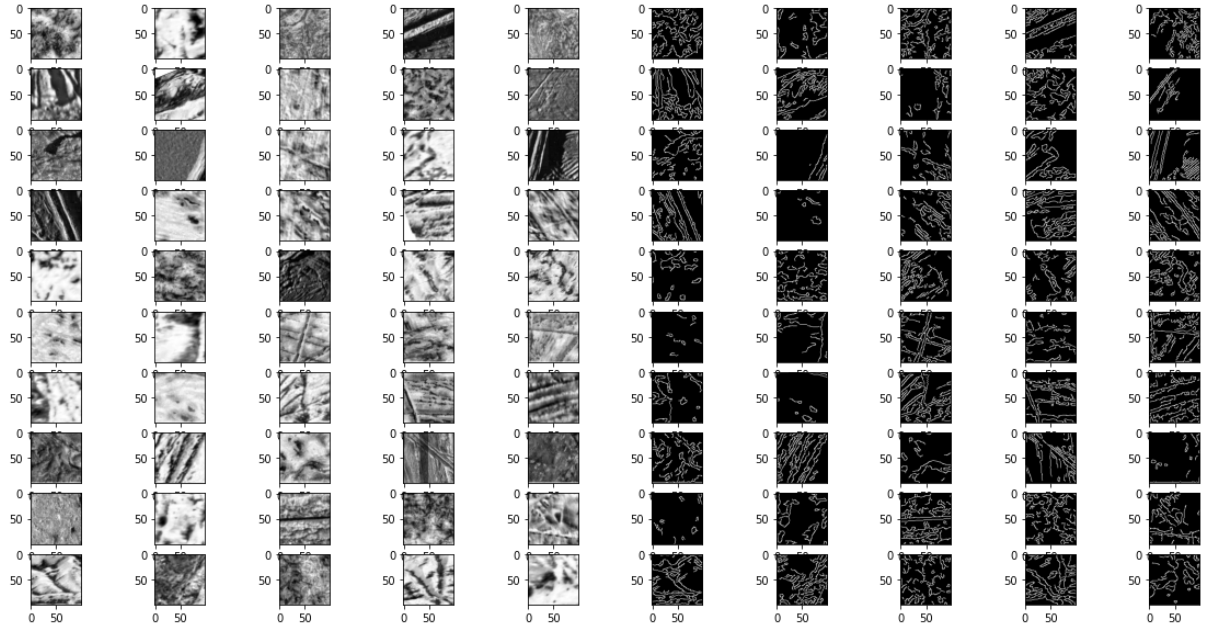


Fig. 26 Some samples of the images that are clustered into the second 'large' group (Group B). The corresponding edges detected from original images are on the right. (See Appendix B for a larger version of the figure and more samples)

Figure 25 and Figure 26 show some samples from the groups. If we interpret this result with human eyes, images that contain complex 'ridges' are more likely to be grouped into Group A, while images with large areas of 'plains' may be clustered into Group B.

Group A

Then Figure 27 indicates that we can further split Group A into two smaller groups, we call the first smaller group **Group a** and the second smaller group **Group a***.

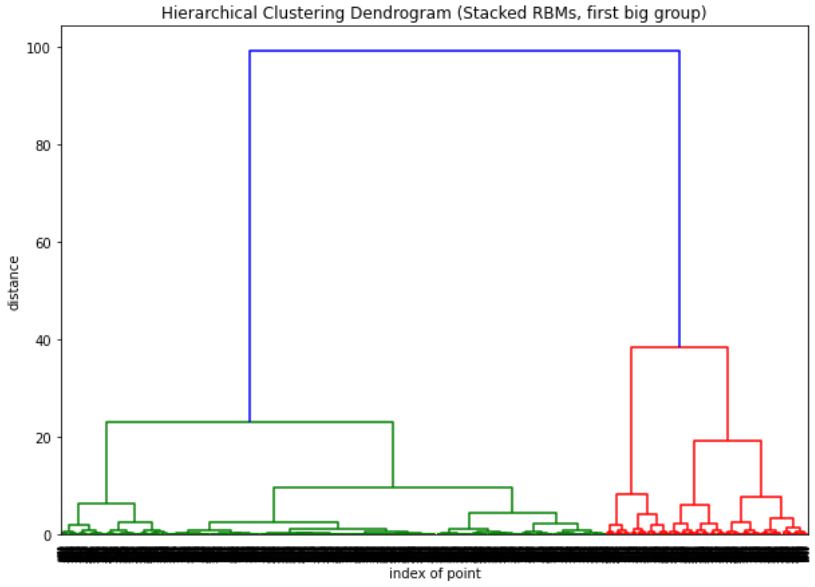


Fig. 27 The dendrogram of Group A. There is a large increase in distance between clusters when the number of clusters equals 2.

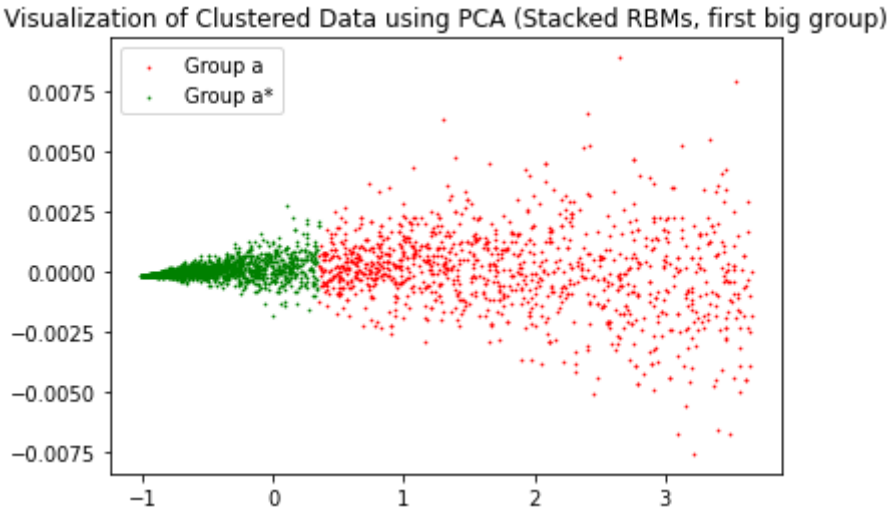


Fig. 28 Visualize the clusters of Group A. Data are projected to two-dimensional space by PCA, and encoded data are grouped using hierarchical clustering algorithms.

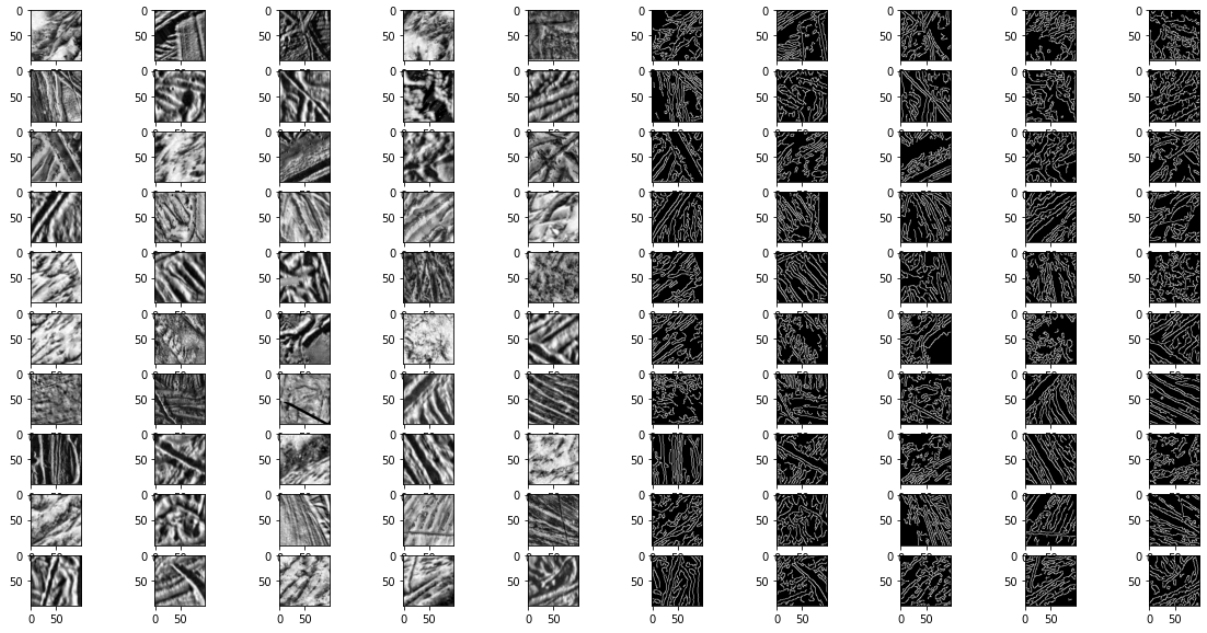


Fig. 29 Some samples of the images that are clustered into Group a. The corresponding edges detected from original images are on the right. (See Appendix C for a larger version of the figure and more samples). Images in Group a have sharper ridges than Group a*.

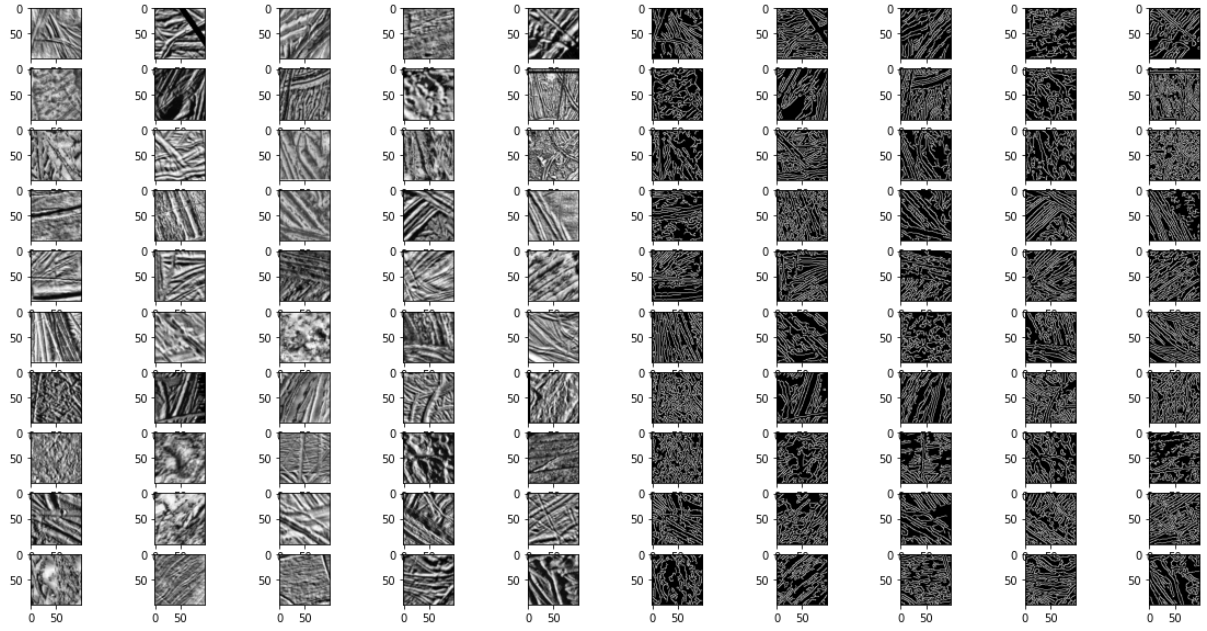


Fig. 30 Some samples of the images that are clustered into Group a*. The corresponding edges detected from original images are on the right. (See Appendix D for a larger version of the figure and more samples)

By comparing Figure 29 and Figure 30, we can notice that most of the images in Group a have deeper and wider ridges than Group a*. 1127 images are clustered into Group a and 3036 images are clustered into Group a*. And Group a* has a smaller average within-cluster distance than Group a (Figure 28).

Group B

The large increase of distance when the number of clusters equals two shows that we can further divide Group B into two smaller groups: **Group b and Group b*** (Figure 31).

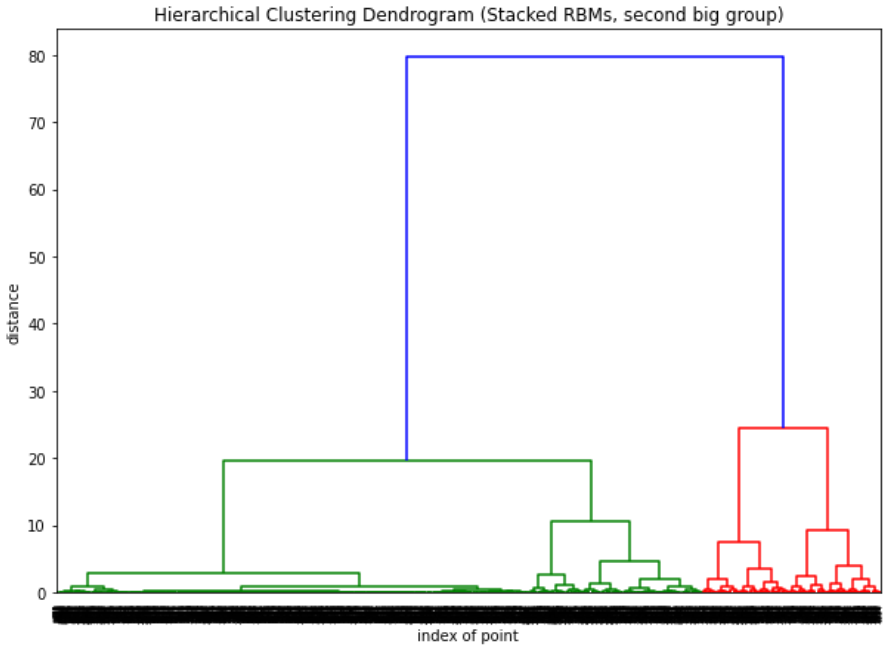


Fig. 31 The dendrogram of Group B. There is a large increase in distance between clusters when the number of clusters equals 2.

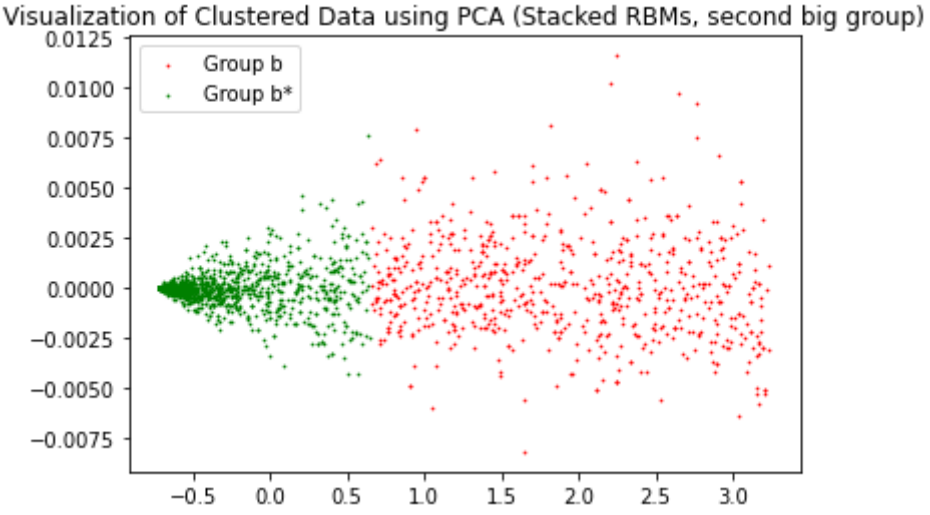


Fig. 32 Visualize the clusters of Group B. Data are projected to two-dimensional space by PCA, and encoded data are grouped using hierarchical clustering algorithms.

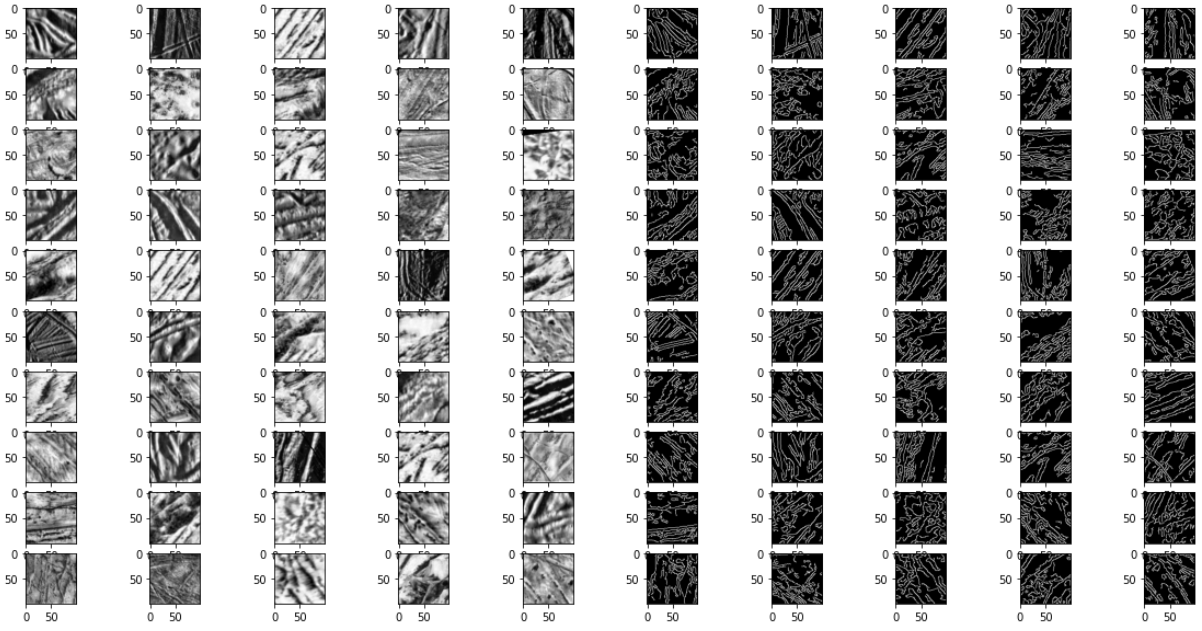


Fig. 33 Some samples of the images that are clustered into Group b. The corresponding edges detected from original images are on the right. (See Appendix E for a larger version of the figure and more samples) 704 images are classified into Group b.

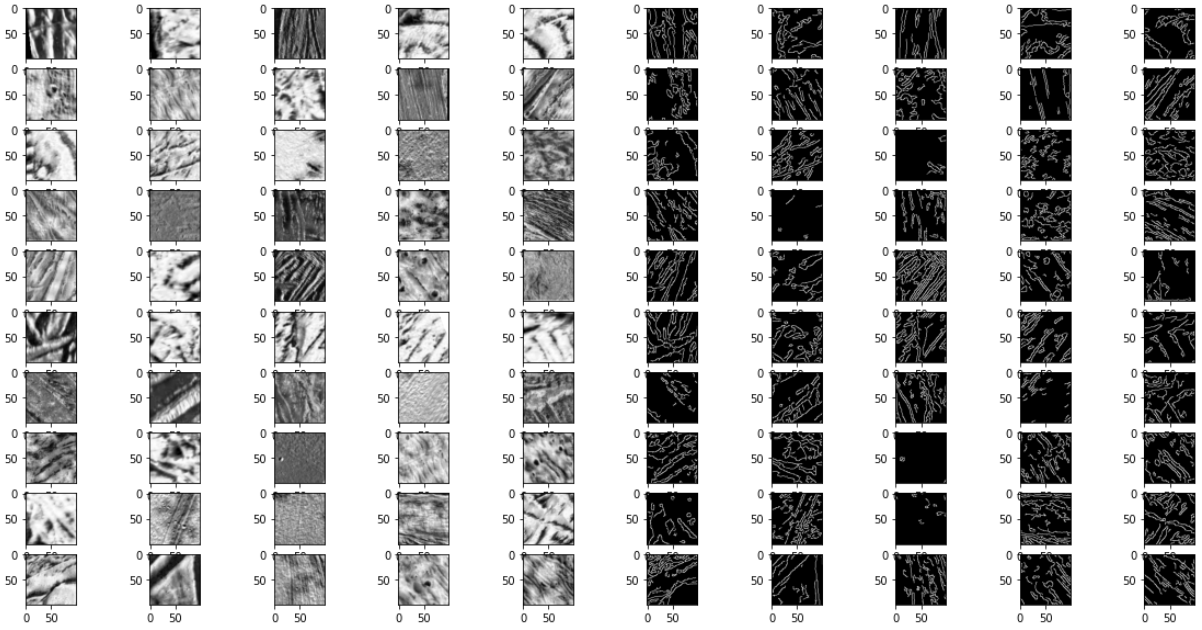


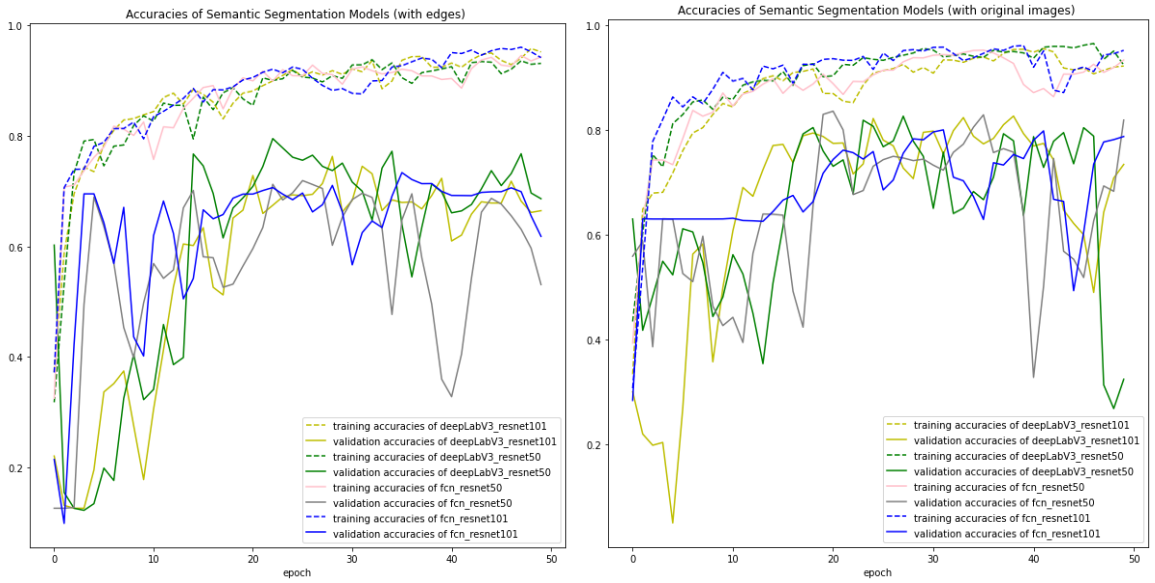
Fig. 34 Some samples of the images that are clustered into Group b*. The corresponding edges detected from original images are on the right. (See Appendix F for a larger version of the figure and more samples) Group b* is the dominant subset of Group B; 2657 images are in Group b*. Images in Group b* look plainer than Group b.

Most of the images in Group b* consists of a large area of plains (Figure 34) and some have a few craters. However, it is hard to interpret the images in Group b, no obvious patterns are found. 704 images are clustered into Group b and 2567 images are clustered into Group b*: Group b* is the dominant subset of Group B, and Group b* has a smaller average

within-cluster distance than Group b (Figure 32). It seems that Group b is the minority that shares some common characteristics with Group b* but is very different from Group A.

To summarize, we divide the images into four clusters: Group a, Group a*, Group b, and Group b*. It seems easy to understand the results when machines filter out images with ridges. However, it is hard to explain, with human eyes, the groupings of images that look plain.

5.2 Semantic Segmentation



(a) (b)

Fig. 35 Accuracies of semantic models during training. (a) Models are trained with edges; (b) models are trained with original images. Both (a) and (b) encounter an overfitting problem. Since we don't have much available data, the validation set here is also used as a test set.

We have 44 images overall for our segmentation tasks. Models are trained with 40 images and the rest are used as the test set. All segmentation models can reach approximately 0.80 accuracies for the test set, and the accuracies trained with original images are higher than those trained with edges. After a comparison of accuracies on the test set, the DeepLabV3-Resnet101 model trained on original images is chosen to do the final segmentation tasks. Figure 36 are the predictions of all images by the model. Figure 37 suggests that it clearly highlights some ridges, band, craters, and chaos on images of the test set, which is a satisfying result.

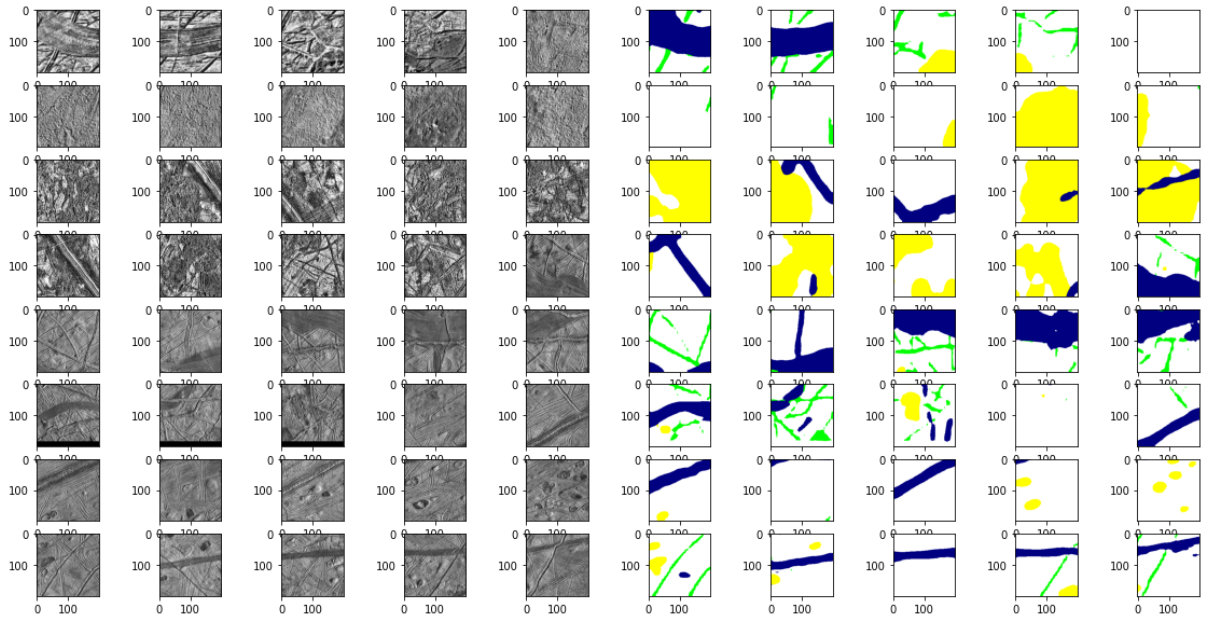


Fig. 36 On the right are the predictions of the corresponding images by DeepLabV3-Resnet101--*White: plain; navy blue: band; green: ridge; yellow: chaos; orange: craters.* (Samples are from both the test set and training set, see Appendix G for a larger version of the figure)

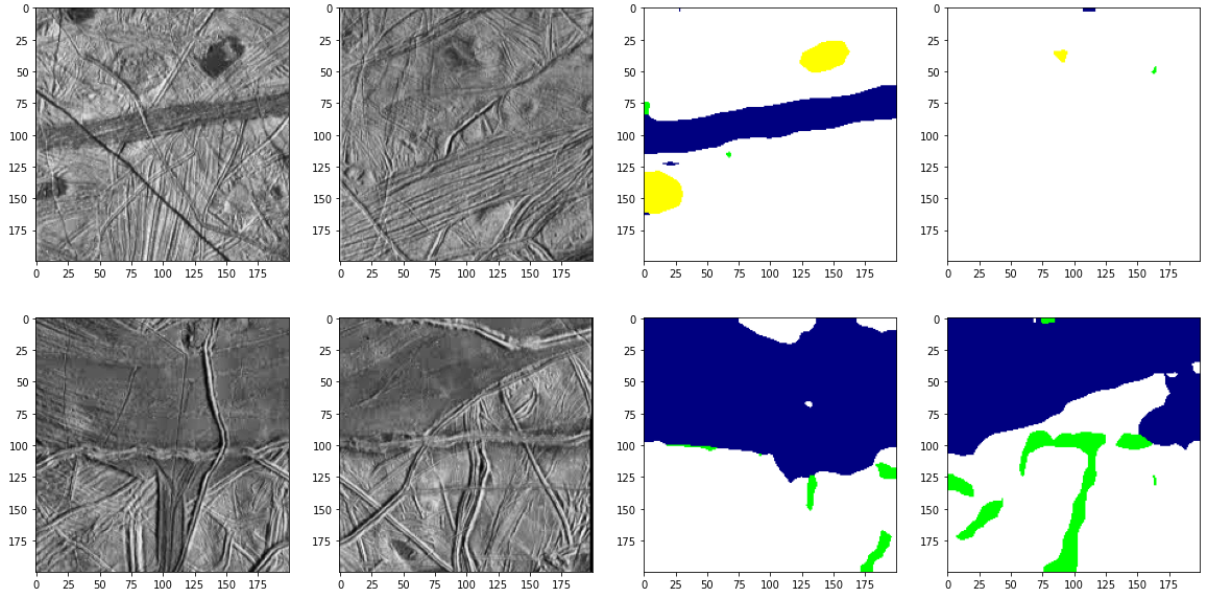


Fig. 37 Predictions by DeepLabV3-Resnet101 (Samples are from test set). *White: plain; navy blue: band; green: ridge; yellow: chaos; orange: craters.*

6. Conclusion and Discussion

Despite being a task for classifying fractures on Europa, this project can also be regarded as a study case for a comparison of various machine learning-based dimension reduction

techniques. For our original image training set, among all the autoencoders, latent representations extracted by denoising and sparse autoencoders are easier to cluster than that from the basic autoencoder with the same structure.

When time is limited, it's hard to develop a relatively 'perfect' autoencoder from scratch for a new dataset. In these cases, the stacked-RBMs is a good starting point for dimension reduction or feature extraction, if training data can be converted into binary inputs. However, training stacked RBMs is time-consuming as it uses a Gibbs sampling algorithm (our training of stacked RBMs costs 4 hours for 100 epochs).

In our project, images of fractures on Europa are divided into four classes: Group a, Group a*, Group b, and Group b*, while the paper *'Analysis of very-high-resolution Galileo images and implications for resurfacing mechanisms on Europa' (Leonard, Pappalardo and Yin, 2018)* identify fracture units into plains, ridges, chaos, craters, and bands. The difference in classification is probably due to the following reasons:

1. As the stacked RBMs only accept binary inputs, the way of edge detection will influence the results of stacked RBMs. In our project, canny edge detection is applied to output edges from images; although canny edge detection has an advantage of dealing with noises, it could still produce no-closed boundaries and spurious edge responses (Lelore and Bouchara 2013). There are other edge detection techniques to be explored, such as Sobel, Prewitt, Roberts, and fuzzy logic methods.

2. Although we have algorithms to crop away the defects from images, some images after cropping still have a small area of a black or white region, which can be misleading when we extract edges.

3. One can find it hard to measure how well the images are classified with unsupervised learning approaches. And the classification by machines is hard to interpret with human eyes.

For semantic segmentation using the transfer learning technique, it is demonstrated that we could still hit remarkably high accuracies (testing accuracies approaching 0.88 by DeepLabV3-Resnet101 model) with only 40 training images. However, this semantic segmentation is fully-supervised, and the labels of the fracture types are based on hypothesis rather than ground truth (the knowledge of Europa is limited). Unsupervised-learning semantic segmentation may help us avoid this issue, but boundaries between objects are often ambiguous in satellite images, unsupervised-learning segmentation can barely obtain better performance than fully-supervised methods.

Future work

For image classification tasks, we can spend more time exploring different edge detection methods and more variants of autoencoders such as GANs to extract features. And as for semantic segmentation tasks, we need to deal with the overfitting problems.

References

Arpit, D. *et al.* (2015) 'Why Regularized Auto-Encoders learn Sparse Representation?' Available at: http://arxiv.org/abs/1505.05561 (Accessed: 27 August 2020).

Beyer, K. *et al.* (1999) 'When Is "Nearest Neighbor" Meaningful?', in. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.1422 (Accessed: 26 August 2020).

Deng, L., Fan, C. and Zeng, Z. (2017) 'A sparse autoencoder-based deep neural network for protein solvent accessibility and contact number prediction', *BMC bioinformatics*. BioMed Central, 18(16), pp. 211–220.

Garg, P. and Jain, T. (2017) 'A Comparative Study on Histogram Equalization and Cumulative Histogram Equalization', *International Journal of New Technology and Research*, 3(9), pp. 41–43.

Goodfellow, I., Bengio, Y. and Courville, A. (2016) *Deep Learning*. MIT Press.

He, K. *et al.* (2015) 'Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification'. Available at: http://arxiv.org/abs/1502.01852 (Accessed: 26 August 2020).

Hartigan, J.A. and Wong, M.A. (1979). Algorithm AS 136: A K-Means Clustering Algorithm. *Applied Statistics*, 28(1), p.100.

Hinton, G. E. and Salakhutdinov, R. R. (2006) 'Reducing the dimensionality of data with neural networks', *Science*, 313(5786), pp. 504–507.

Ketchen Jr., D.J. and Shook, C.L. (1996). THE APPLICATION OF CLUSTER ANALYSIS IN STRATEGIC MANAGEMENT RESEARCH: AN ANALYSIS AND CRITIQUE. *Strategic Management Journal*, 17(6), pp.441–458.

Leonard, E.J., Pappalardo, R.T. and Yin, A. (2018). Analysis of very-high-resolution Galileo images and implications for resurfacing mechanisms on Europa. *Icarus*, 312, pp.100–120.

Ioffe, S. and Szegedy, C. (2015) 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. Available at: http://arxiv.org/abs/1502.03167 (Accessed: 26 August 2020).

Minaee, S. *et al.* (2020) 'Image Segmentation Using Deep Learning: A Survey'. Available at: http://arxiv.org/abs/2001.05566 (Accessed: 26 August 2020).

Pedregosa, F. *et al.* (2011) 'Scikit-learn: Machine Learning in Python', *Journal of machine learning research: JMLR*, 12(85), pp. 2825–2830.

Rokach, L. and Maimon, O. (2005). Clustering Methods. *Data Mining and Knowledge Discovery Handbook*, pp.321–352.

R, R. *et al.* (2019) ‘A Review on Edge detection Technique “Canny Edge Detection”’, *International Journal of Computer Applications*, pp. 28–30. doi: 10.5120/ijca2019918828.

Srinidhi, C. L., Ciga, O. and Martel, A. L. (2019) ‘Deep neural network models for computational histopathology: A survey’. Available at: http://arxiv.org/abs/1912.12378 (Accessed: 26 August 2020).

Srivastava, N. *et al.* (2014) ‘Dropout: A Simple Way to Prevent Neural Networks from Overfitting’, *Journal of machine learning research: JMLR*, 15(56), pp. 1929–1958.

Sugar, C.A. and James, G.M. (2003). Finding the Number of Clusters in a Dataset. *Journal of the American Statistical Association*, 98(463), pp.750–763.

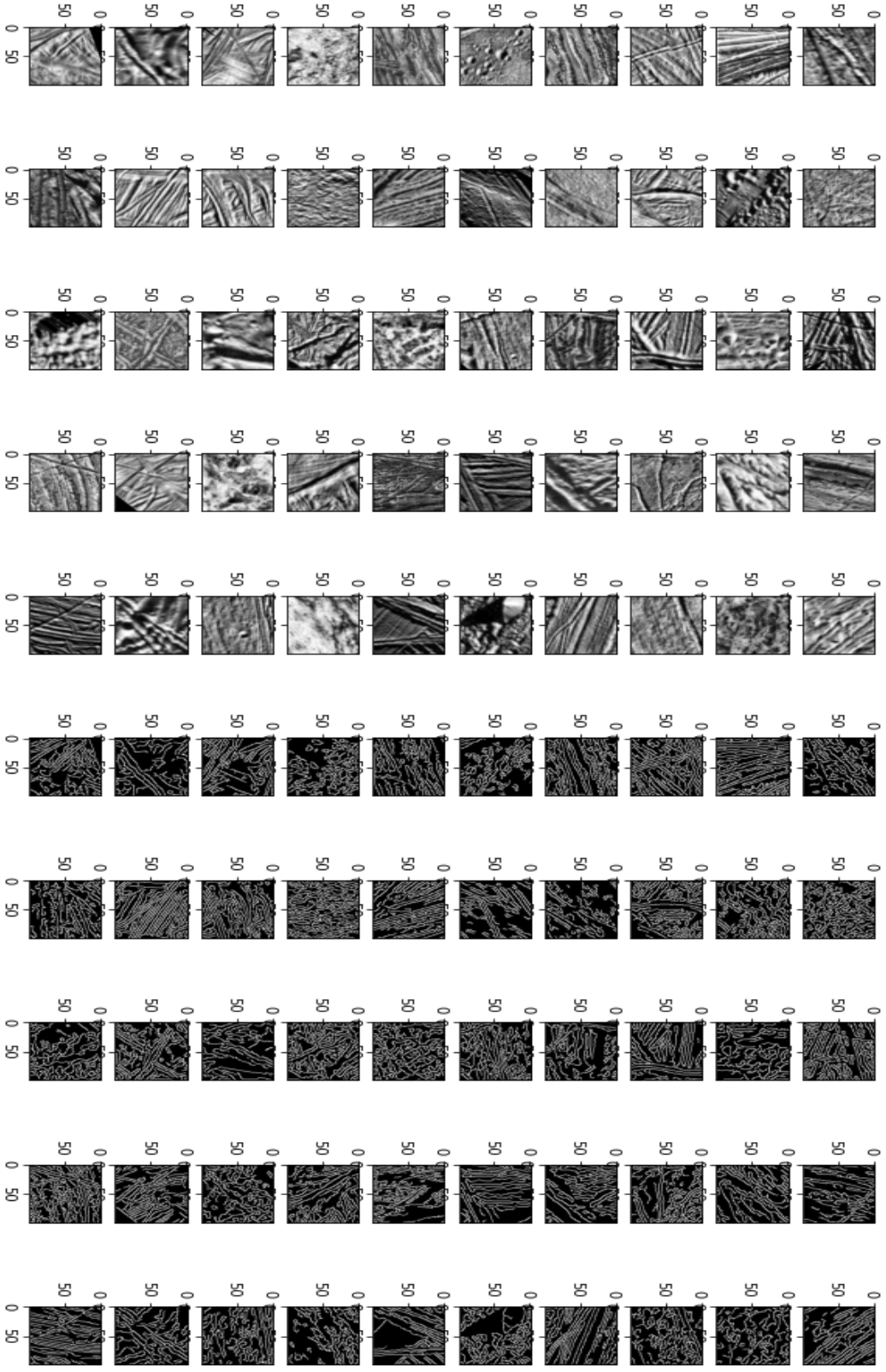
T. Lelore and F. Bouchara, "FAIR: A Fast Algorithm for Document Image Restoration," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 2039-2048, Aug. 2013, doi: 10.1109/TPAMI.2013.63.

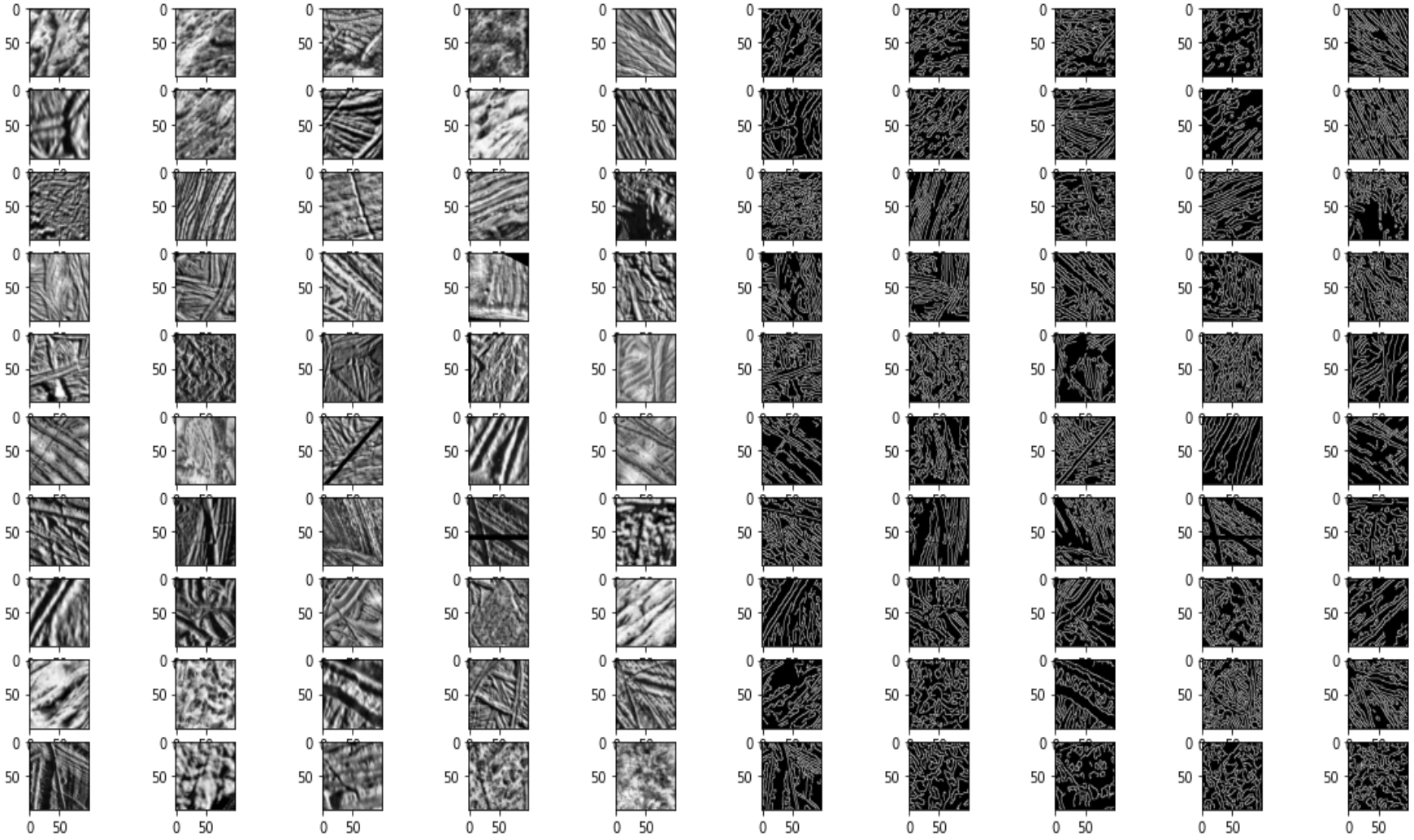
Tieleman, T. (2008) ‘Training restricted Boltzmann machines using approximations to the likelihood gradient’, *Proceedings of the 25th international conference on Machine learning - ICML ’08*. doi: 10.1145/1390156.1390290.

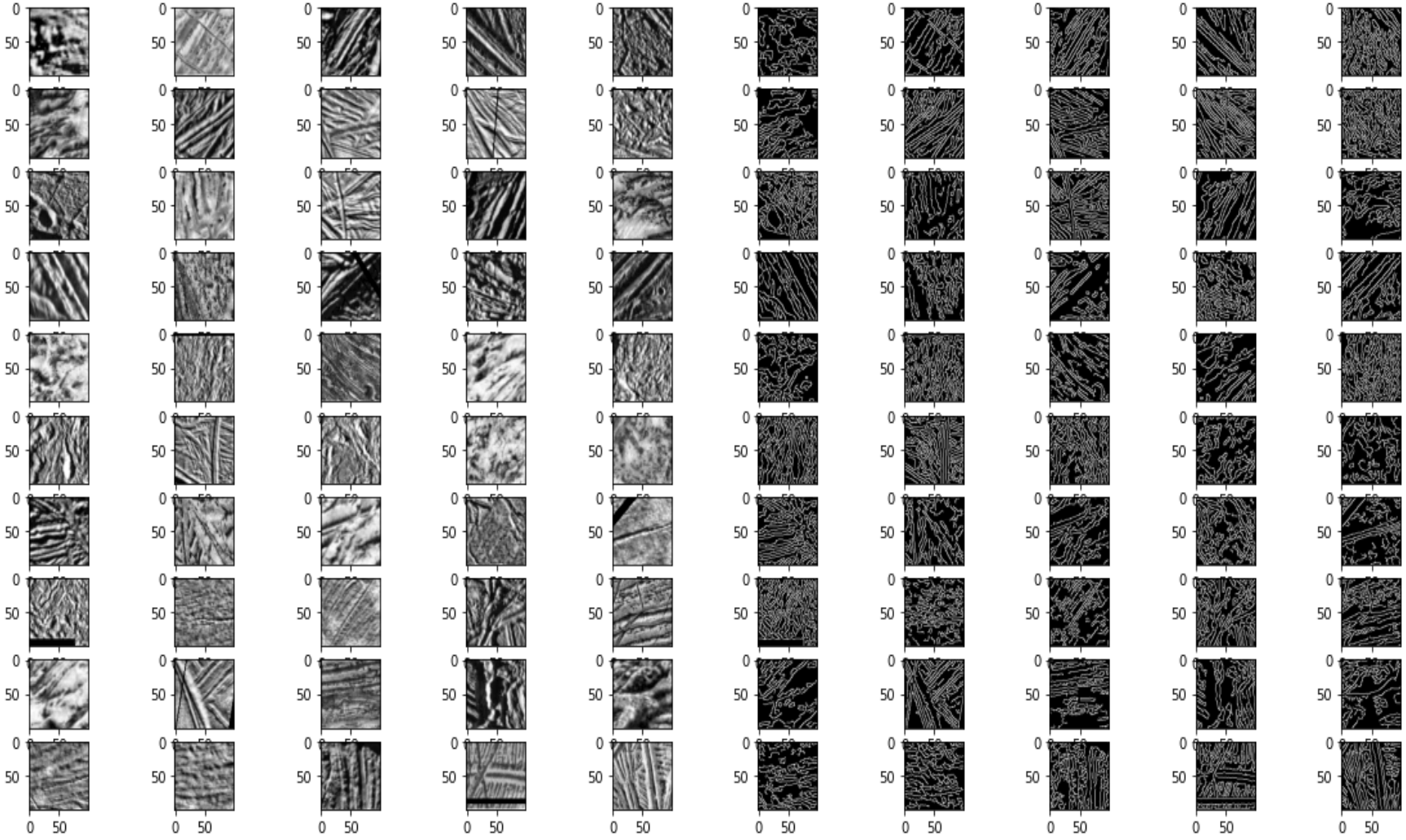
Zuiderveld, K. (1994) ‘Contrast Limited Adaptive Histogram Equalization’, *Graphics Gems*, pp. 474–485. doi: 10.1016/b978-0-12-336156-1.50061-6.

Appendix

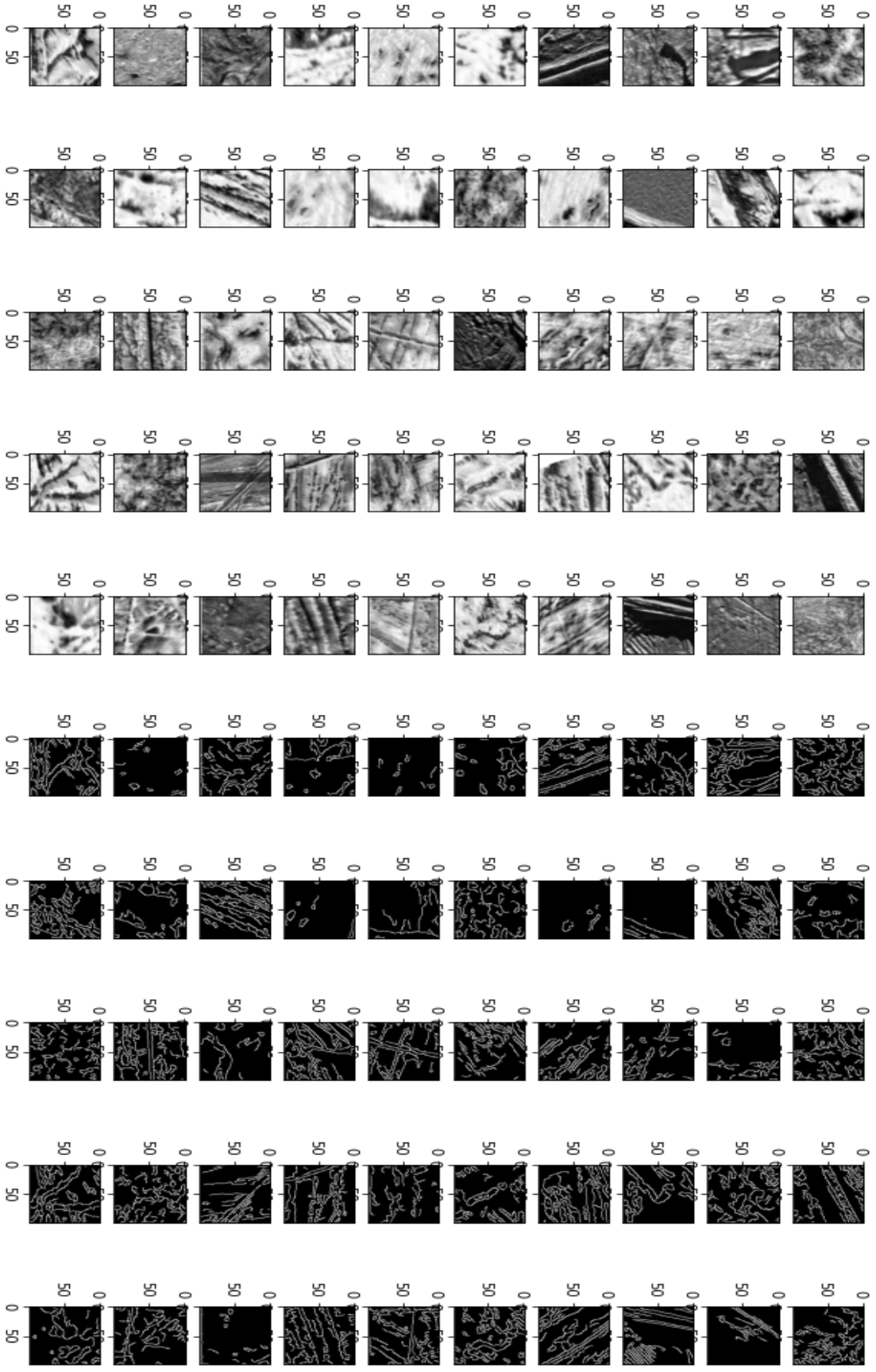
Appendix A

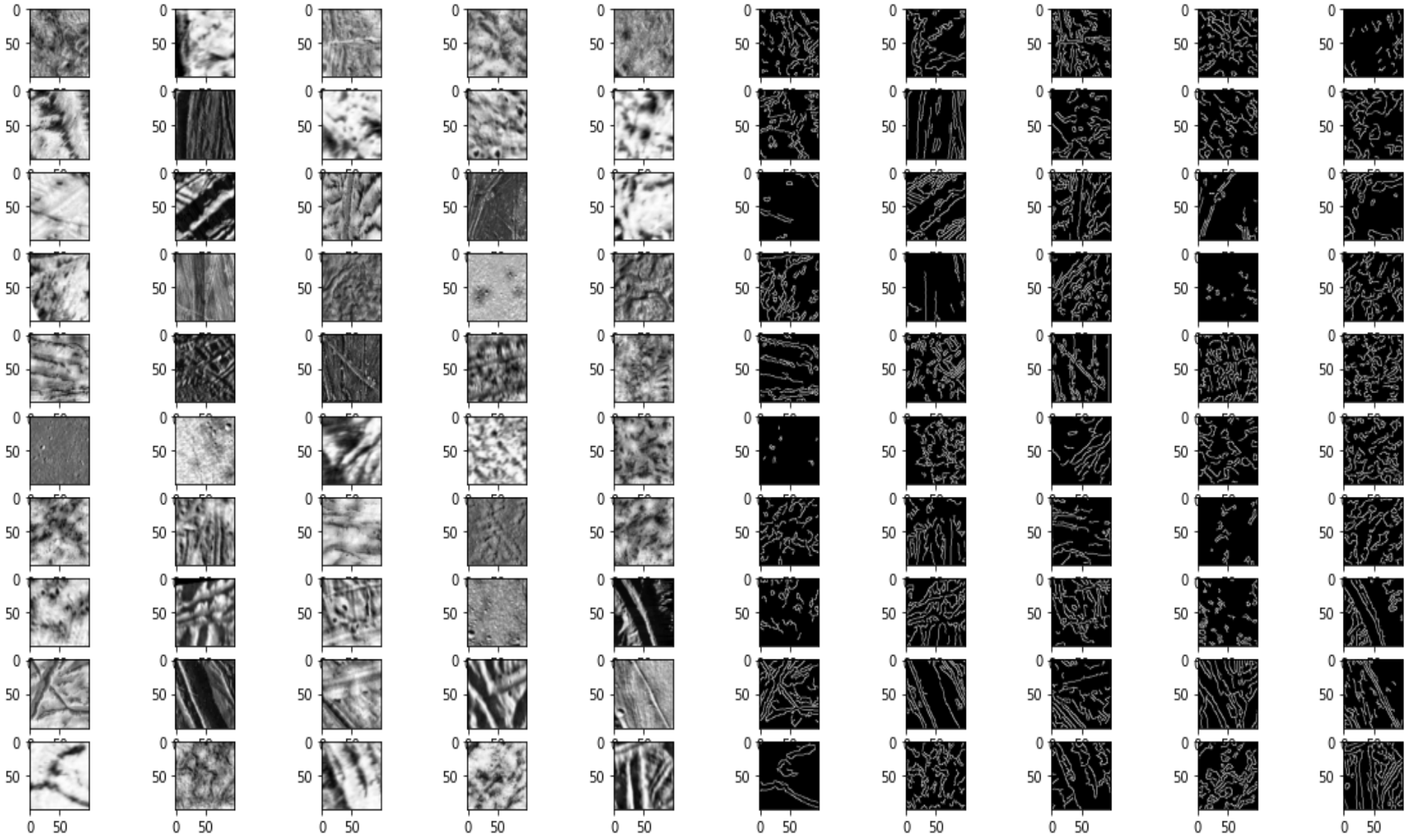


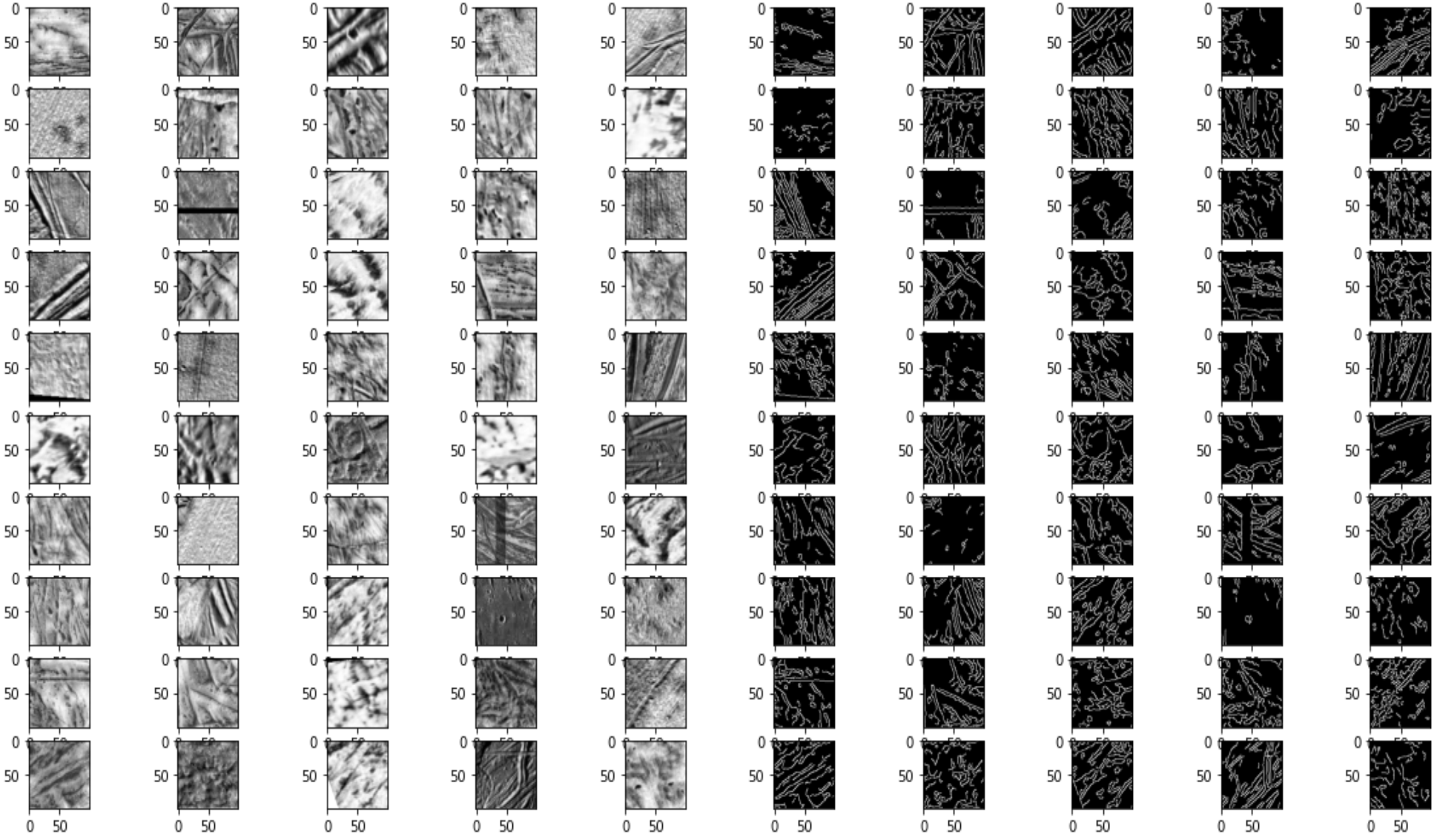

0
50
0 50


0
50
0 50

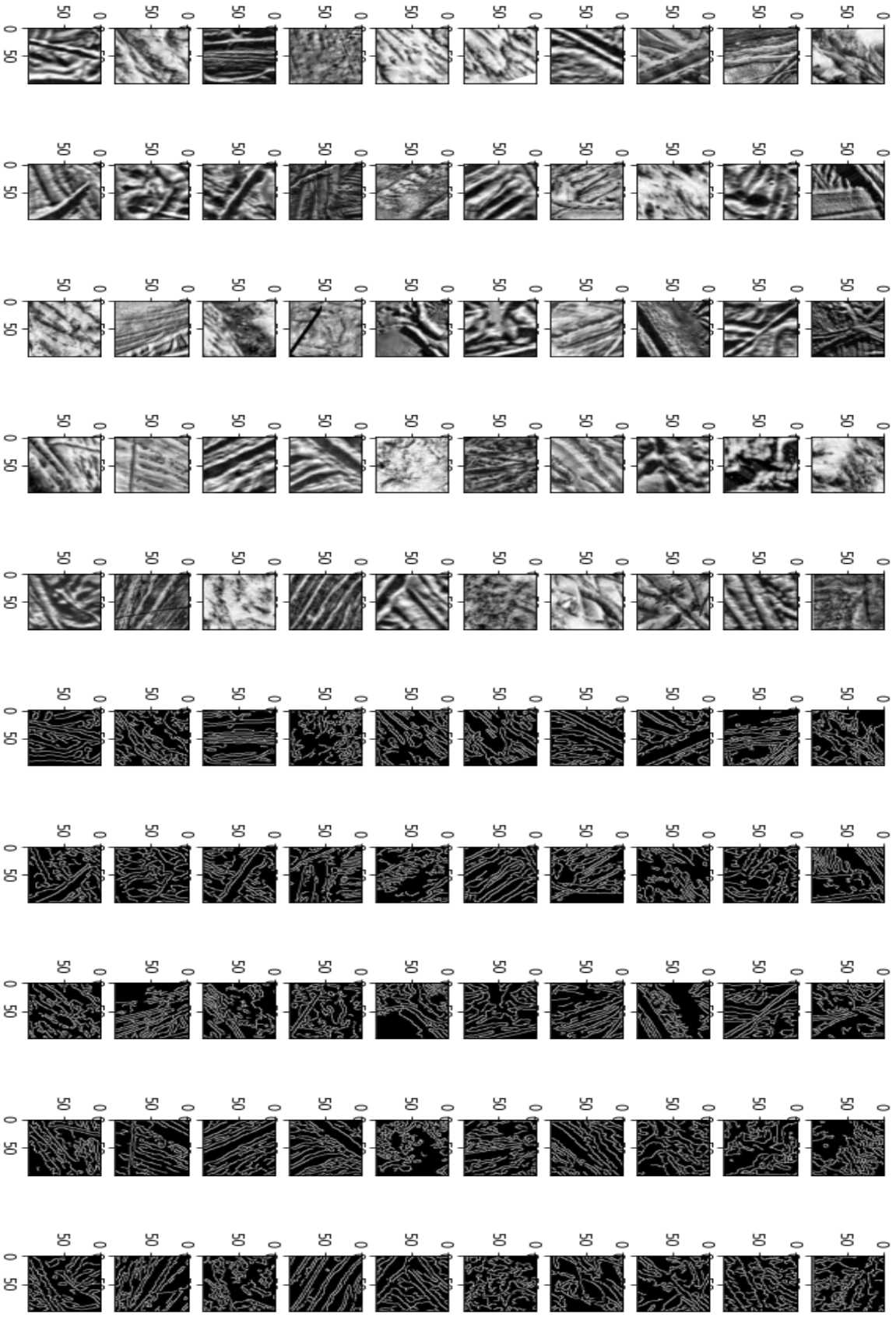
Appendix B

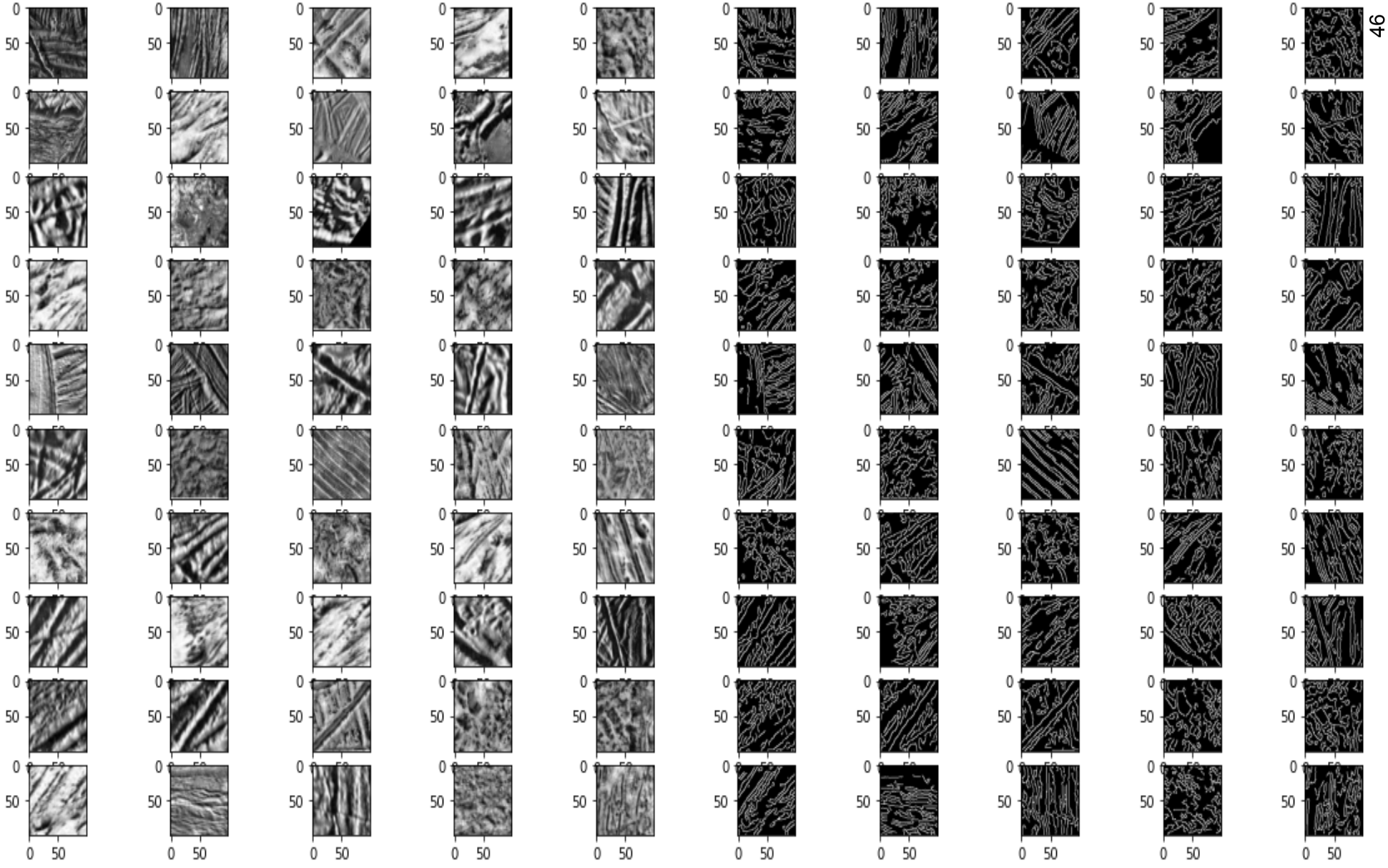


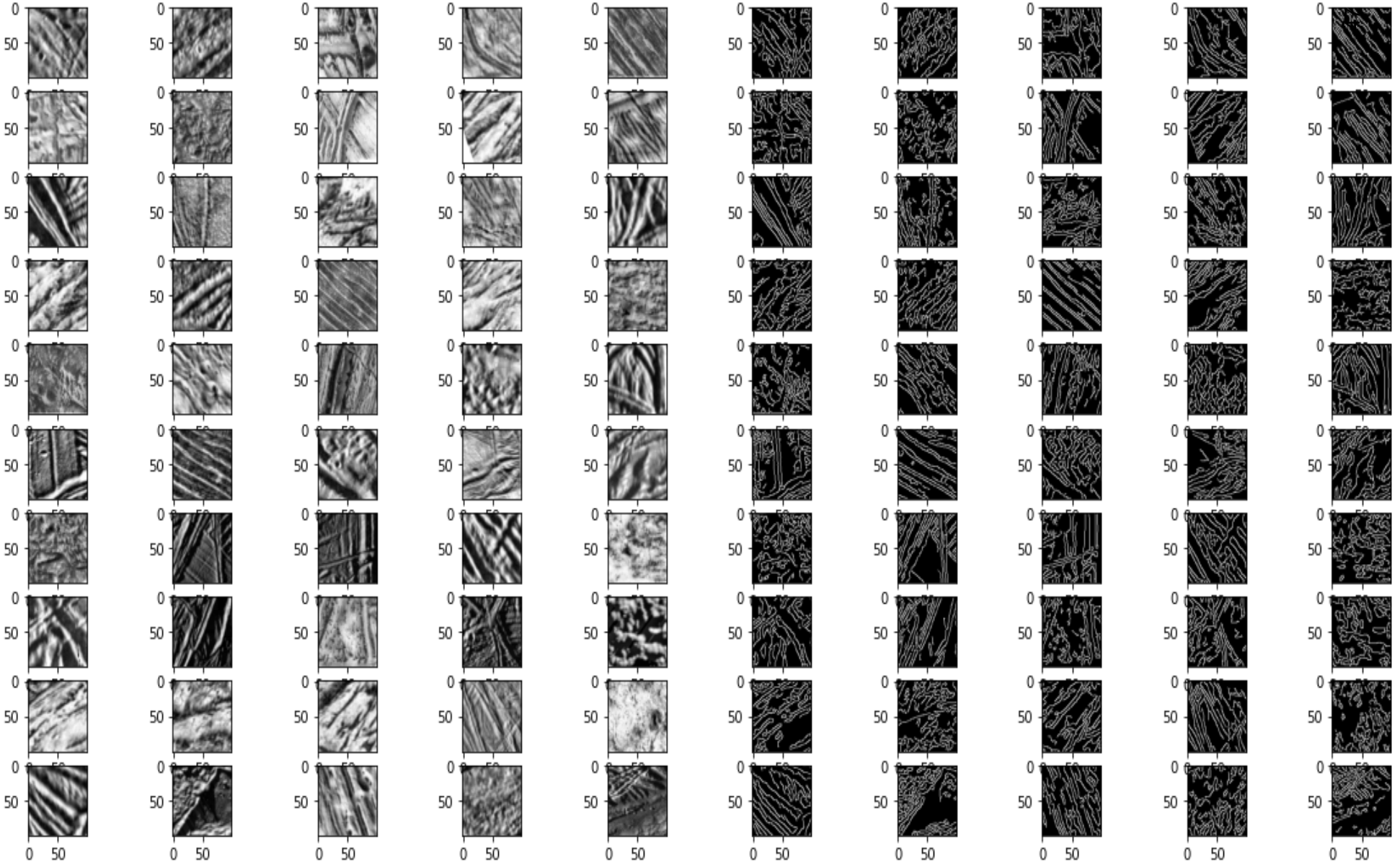

0
50
0 50


0
50

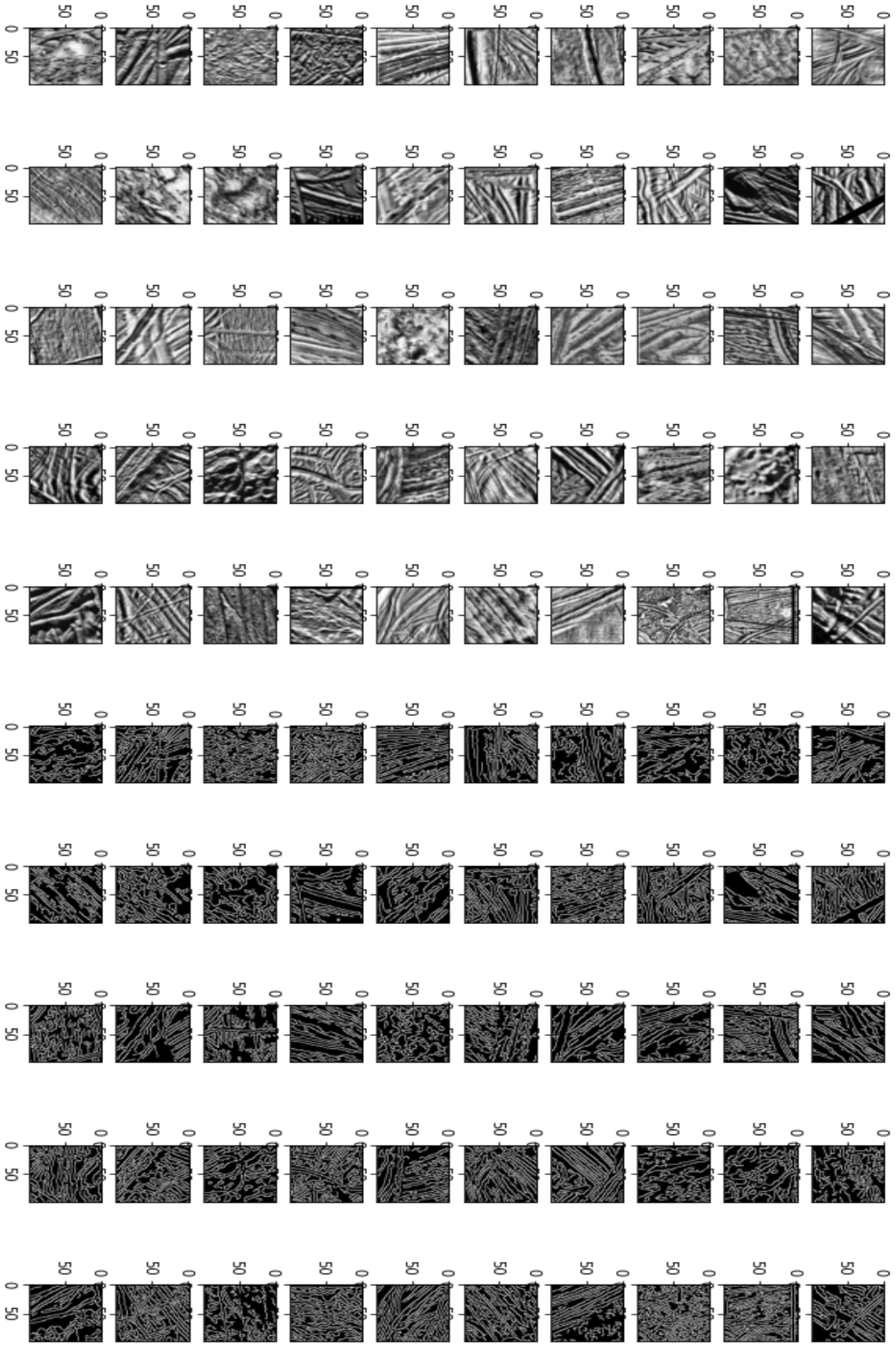
Appendix C

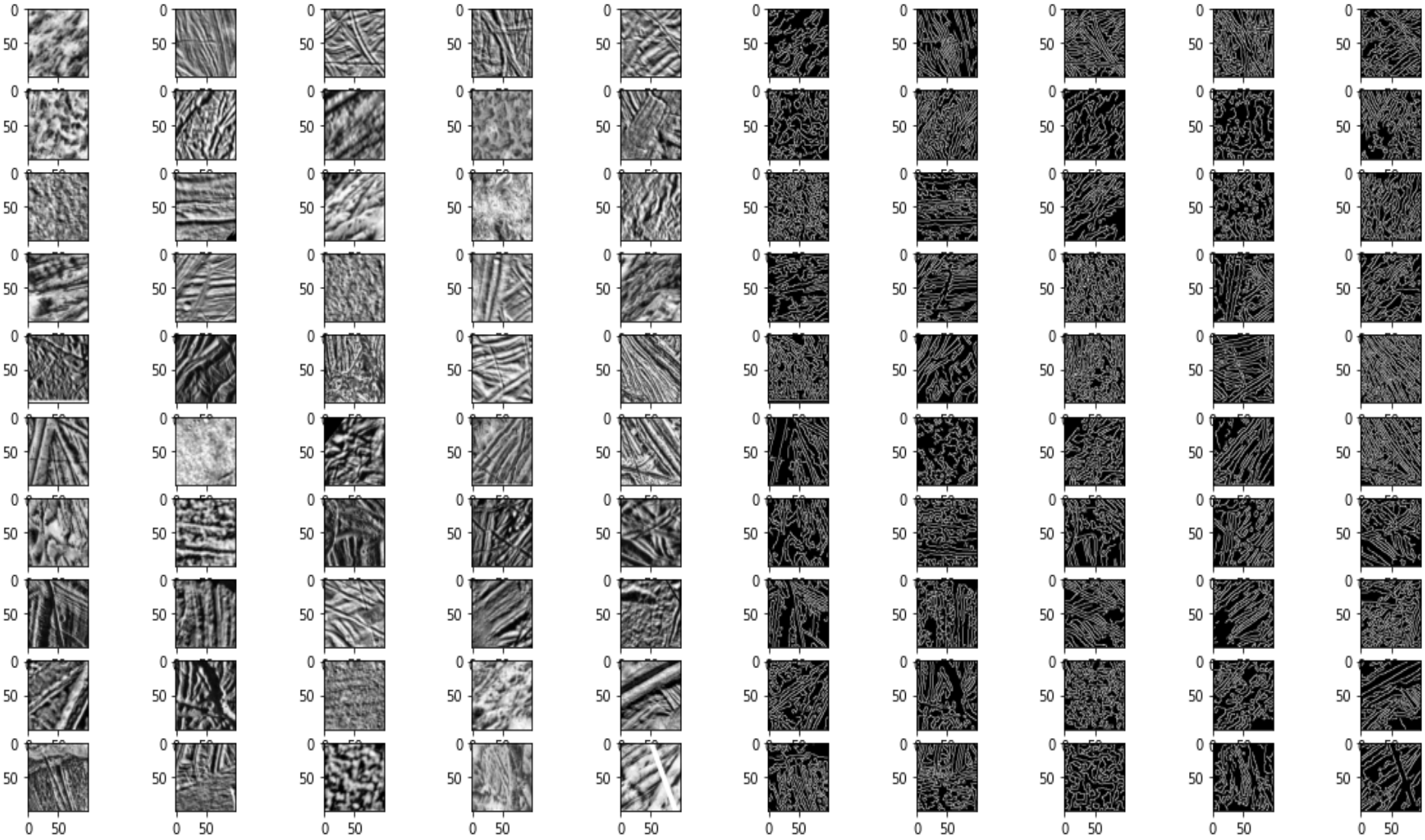


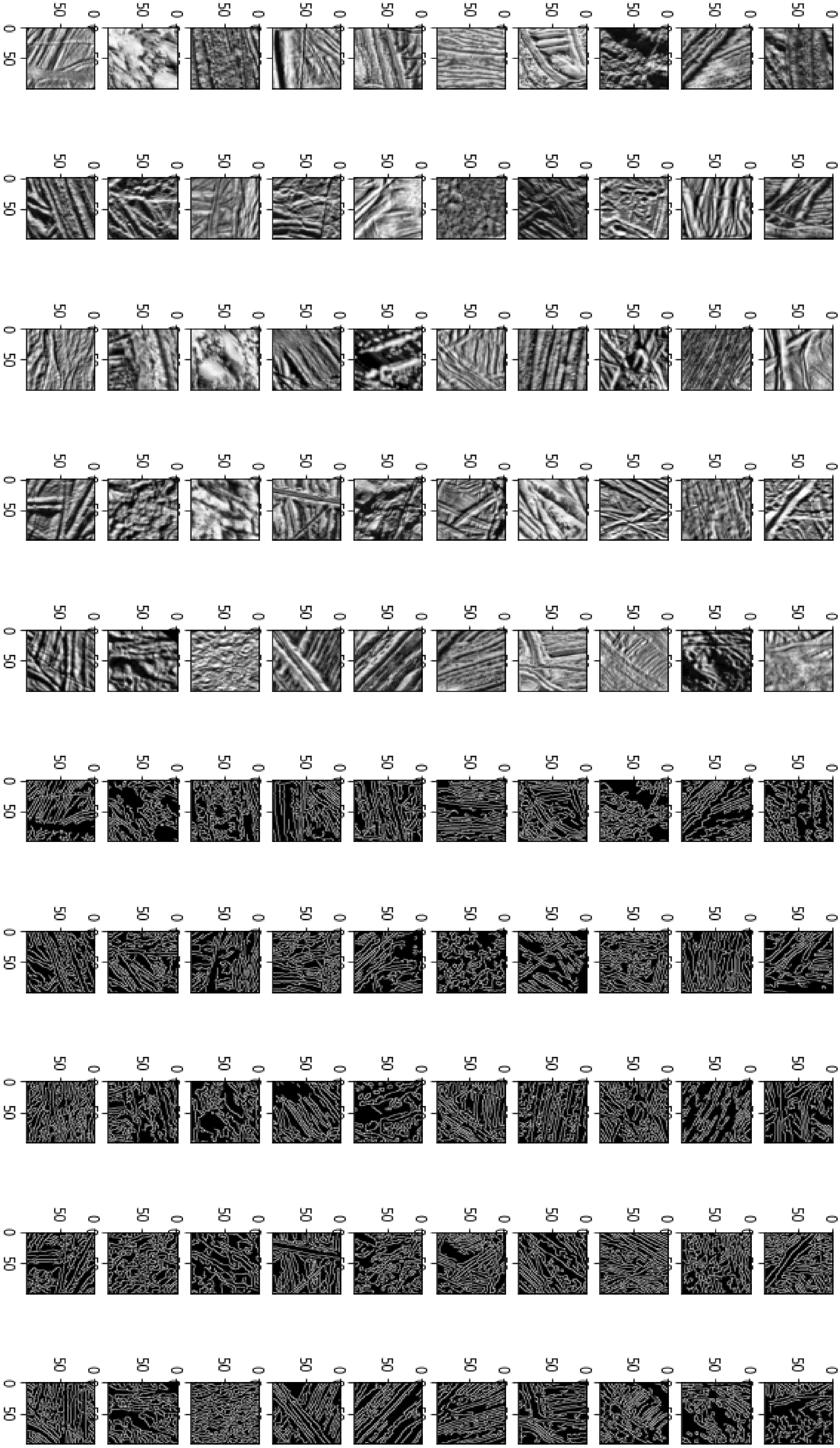

0
50


0
50

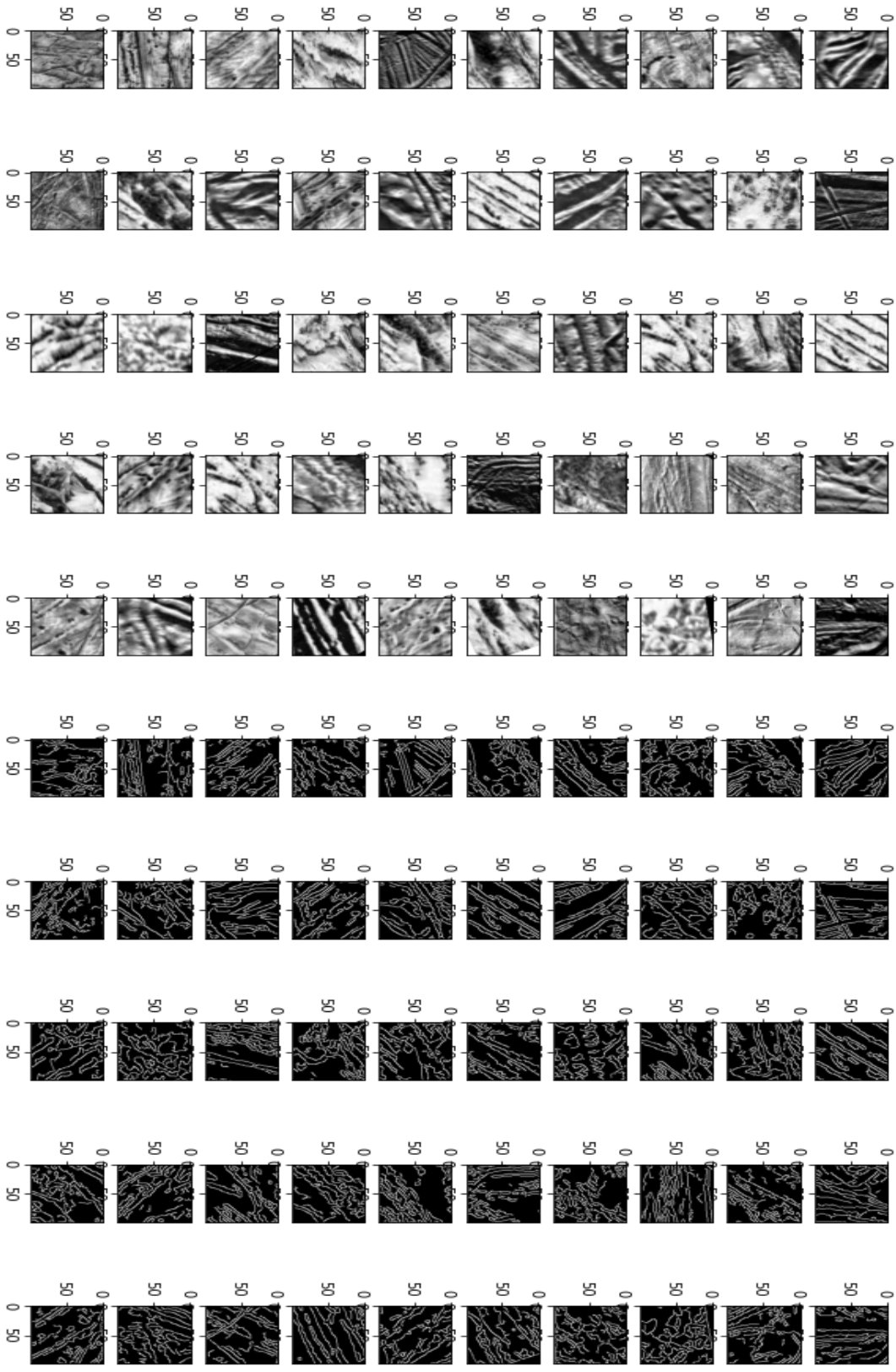
Appendix D

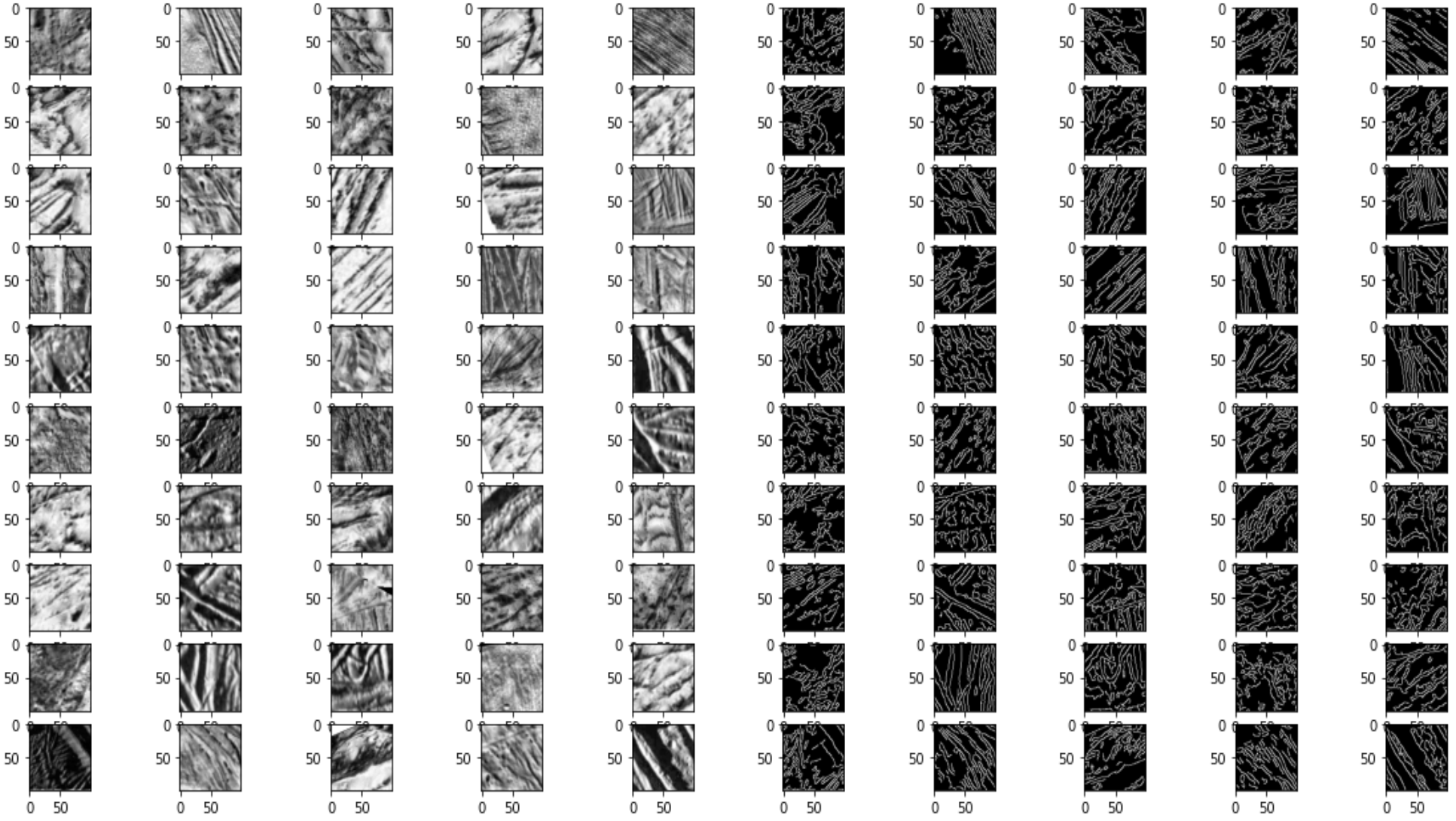



0
50

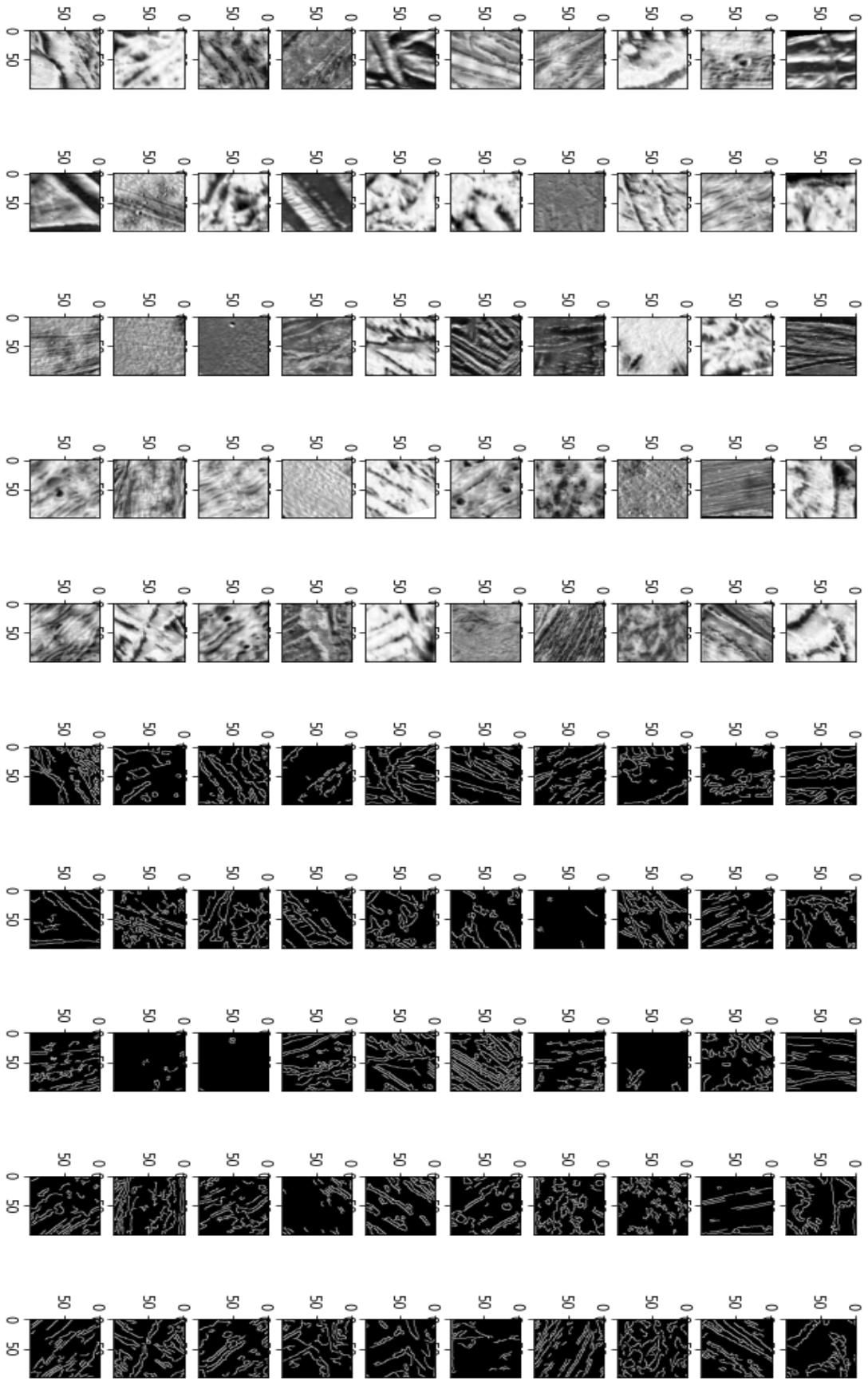


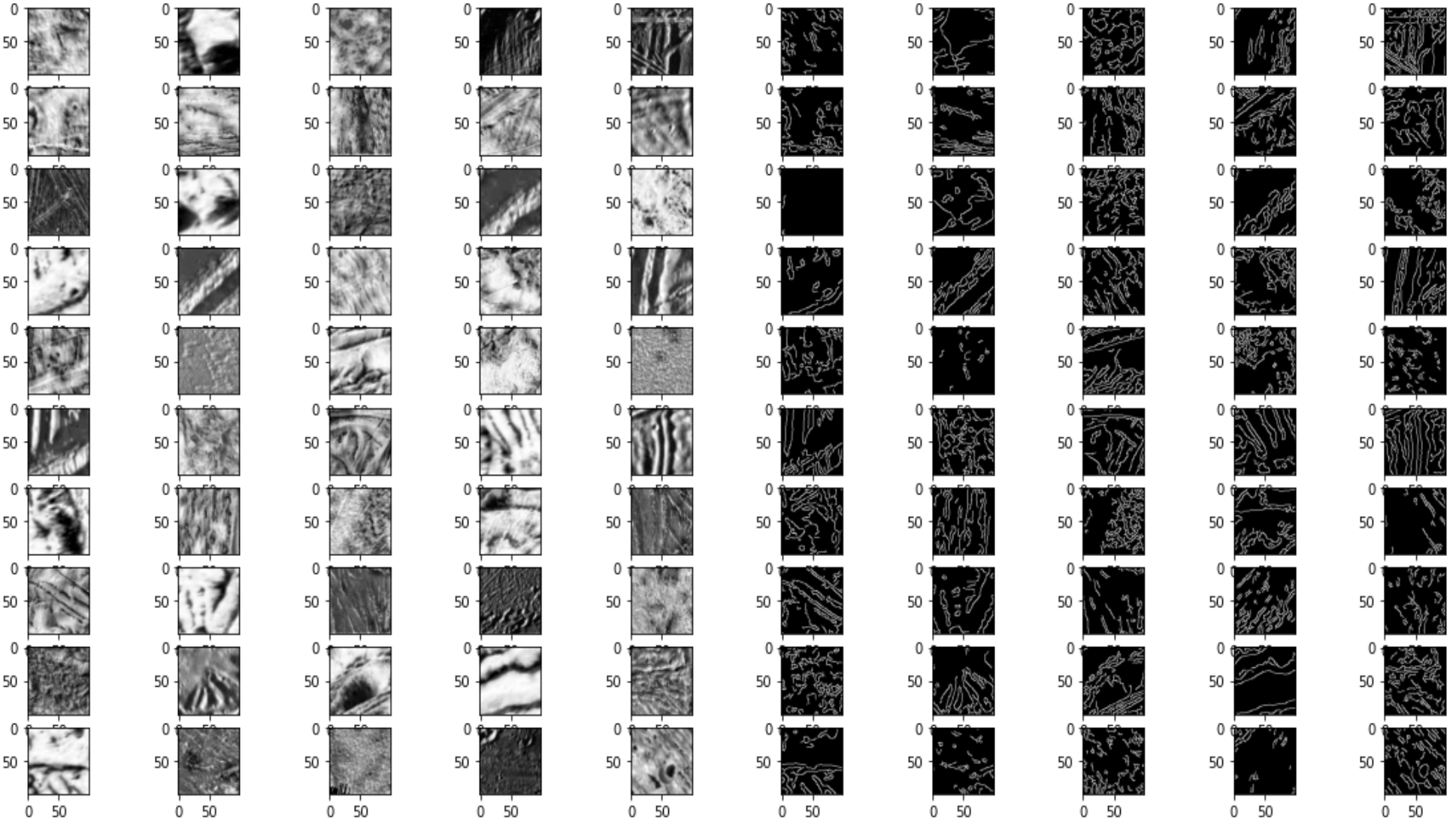
Appendix E

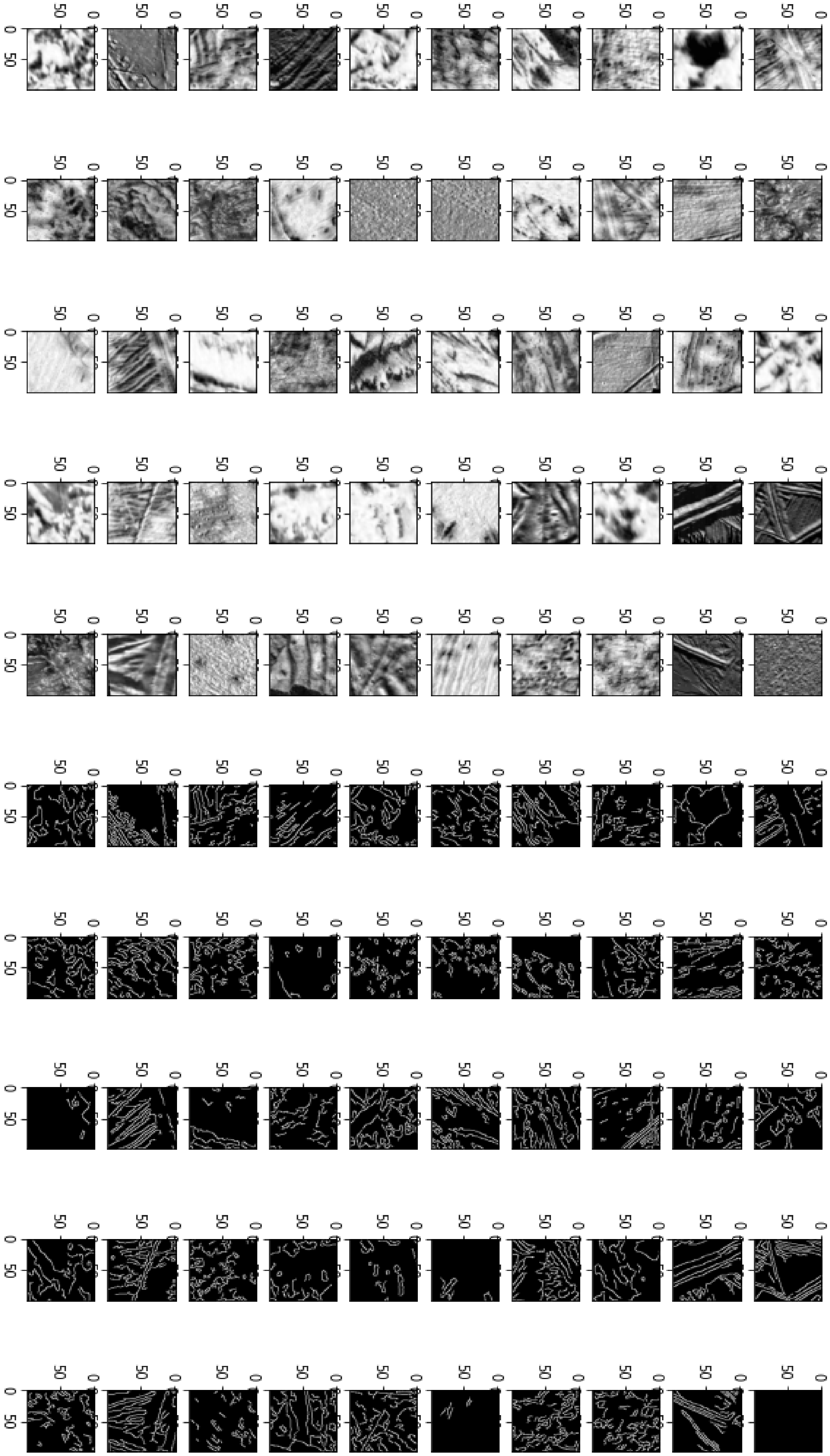



0
50
0 50

Appendix F




0
50
0 50



Appendix G