Institut für Umweltphysik

Master Thesis

Detection and Statistical Analysis of Air-bubble Inclusions in Arctic Sea Ice

Mohammad Ashraful Alam

June 26, 2019

1st Examiner:
Prof. Dr.-Ing. Kai Michels

2nd Examiner:
Dr. Gunnar Spreen

Copyright Declaration

I hereby confirm on my honour that I personally prepared the present academic work and carried out myself the activities directly involved with it. I also confirm that I have used no resources other than those declared. All formulations and concepts adopted literally or in their essential content from printed, unprinted or Internet sources have been cited according to the rules for academic work and identified by means of footnotes or other precise indications of source.

The support provided during the work, including significant assistance from my supervisor has been indicated in full.

The academic work has not been submitted to any other examination authority. The work is submitted in printed and electronic form. I confirmthat the content of the digital version is completely identical to that of the printed version.

Date: _____ Signature: _____

Acknowledgement

Before I start writing this report, I would like to convey my deep respect to my tutor and thesis supervisor Dr. Gunnar Spreen to give me this thesis topic. I would also like to express my gratitude for his help, guidelines and time to time monitoring. Without his monitoring and guidelines, it would not be possible to accomplish this thesis successfully in time. Alongside, I would like to thank my main supervisor Prof. Dr.-Ing. Kai Michels to allow me to do my master thesis in "Institut für Umweltphysik" department. I would also like to extend my gratitude to Philip Rostosky, Marcus Huntemann, Dimitri Murashkin, and Valentin Ludwig for their continuous help in group discussion and library installation.

The sea ice core images for thick sections analyzed during this research work are provided by the Norwegian Polar Institute. I would like to convey special thanks to all participants from the N-ICE2015 project who prepared the thick section ice core images and in particular to Sebastian Gerland from the Norwegian Polar Institute for providing the data.

Finally, I thank almighty, parents, siblings and friends for their continuous inspiration without which this nice completion would not be possible.

Abstract

The occurrence of the air bubble and brine channel phenomenon is observed in Arctic sea ice. Air bubbles in the ice scatter microwaves and therefore influence the ice emissivity, which results in an influence on the microwave radiometer and radar satellite signals in the range from 1 GHz to 300 GHz over sea ice. In order to investigate these effects, a Convolutional Neural Network (CNN) model is trained using TensorFlow (a group of open source libraries for Dataflow programming) to independently detect the position of air bubbles and brine channels in the image of ice cores. Then OpenCV libraries (a number of open source libraries for computer vision programming) and Matlab image processing tools are simultaneously used to derive the area and circularity of the air bubbles. The frequency of air bubbles is plotted using Python libraries. Jupyter Notebook and PyCharm IDE are used to run the Python scripts.

The trained model is analyzed on 8 different ice cores and in total 50 images. The detection results show that the frequency of air bubble occurrence, their area and the circularity of these are independent of the depth of the ice core within the Arctic Sea area. The number of air bubbles in different images of a single core does not vary significantly with respect to the depth. It is observed that the occurrence of air bubbles having the area above 0.5 mm^2 decreases with the increase of core depth, which also means that the occurrence of smaller air bubbles increases with depth. The performance of the model developed in this process is tested for air bubble detection on 5 different images of a single core and measured in terms of Precision Rate [18]. An average of 86.47% of Precision Rate is obtained from this evaluation.

Contents

1. Introduction

1.1. Background

In the era of Computer Vision and Machine Learning technology, object detection has become a demanding and challenging task throughout the world. Right now, object detection using machine learning algorithms like a classifier, Convolutional Neural Network (CNN), Deep Learning Neural Network (DNN), etc. are frequently being used in many applications like robotics, image processing, image segmentation, image retrieval, and video surveillance.

There are a good number of algorithms already developed for local feature key points detection. All off the algorithms are different from each other based on their methodologies. CNN model is practically developed based on the visual system structure. At first, in 1962, Hubel and Wiesel [16] in their research work on the visual cortex of cat and monkey, showed that cortexes contain cells or neurons that are sensitive to small space of the visual field. The subregion or small space of the visual field, which creates stimuli effect on the cortex neuron, is known as the receptive field [24]. All neighbouring neurons also provide overlapping but either of similar or of different sized receptive field and thereby all receptive regions systematically develop a complete map that helps to identify the objects.

Inspired by the work of Hubel and Wiesel, in 1980, Kunihiko Fukushima [21] introduced a multilayered neural network model called Neocognitron which consists of convolutional layers and downsampling layers. The convolutional layer contains neuron or cell which is often called filter, which is nothing else but only a weight vector consist of a set of adaptive weights. The receptive field of each neuron of the convolutional layer covers an area of the previous convolutional layer and followed by the units or cells of the downsampling layer whose receptive field covers areas of the previous convolutional layer. This downsampling layer helps to classify the object correctly. Neocognitron was the first proposed artificial neural network model which was simulated on computer [16] and applied for handwritten character recognition [9].

Later, after a successful contribution to the research field based on Neocognitron, the CNN model is further developed. Nowadays, CNN or DNN model is frequently used to traffic sign detection [30], pet animal detection [17], vehicle detection [20] and other pattern detection and so on.

1.2. Objective and Motivation

Several studies over the thin sections of the top 300 mm layer of first-year and multi-year ice are conducted to understand the statistics on air bubble dimension and geometry. Study shows that air bubbles in first-year ice (FYI), especially near the surface, are arbitrarily distributed and connected with each other randomly. Satellite sensors operating in the microwave range are the primary tool to monitor the global sea ice cover since the 1970s. The microwave radiometer signal and radar satellite signal over sea ice get influenced due to the alternation of air bubble inclusion range in microwave scale (from 1 GHz to 300 GHz) in ice emissivity and scattering [34].

In order to investigate these effects, simulations of the microwave emission of Arctic sea ice is needed, which requires detailed information about the air bubble's size, circularity, and distribution. Air bubble inclusions can be observed in figure 3.4.

The main goal of this thesis is to develop a model using OpenCV libraries, Matlab and Python programming language which independently can detect and localize air-bubbles and brine channels from a given input image. Furthermore, the model analyzes statistical information from air-bubble and brine channels. Statistical information includes position, area, circularity, the total number of bubbles or channels, size in millimeter, etc. The statistical information is further used to calculate the distribution of air bubbles with respect to the depth of ice cores and plot Probability Density Function (PDF).

Tensorflow (TF) is a framework designed for machine learning dataflow, which is used to design a Convolutional Neural Network for the object (i.e. bubble and channel) recognition, classification, and regression. Besides Tensorflow, a good number of Python-based open source frameworks for the implementation of Artificial Intelligence (AI) and Convolutional Neural Network are already developed. Amongst them, Keras, Theano, Torch, Caffe, YOLO, Microsoft Cognitive Toolkit (Microsoft CNTK) and scikit-learn are popular.

A large number of dataset or samples (training image set and test image set)

is used to train a Tensorflow model. With the dataset, the model is trained till a satisfactory number of iterations. The number of iteration and the changes in losses with iterations is visualized in a graph. When the losses reach a certain level, the training process is stopped immediately. After being properly trained and tested, the Tensorflow model is applied to an image for evaluation. The model provides a number of rectangles, where each rectangle contains air bubble or brine channel individually.

Usually, an air bubble is of circular or elliptical shape and a channel is elongated. Therefore, an object does not logically cover the whole rectangle area solely; rather the rest part of the rectangle is covered by the image background. Once a rectangle is determined, it becomes easier to derive the pure object from the rectangle using OpenCV and Matlab. OpenCV and Matlab have plenty of tools that help us to bring out the whole object from the rectangle area.

Area refers to the occupied number of pixels in an image of an object. Then the number of pixels multiplied by the area of each pixel in unit millimetre square (mm^2) returns the total area in millimetre square (mm^2).

2. Theoretical Background

2.1. CNN Architecture

Mathematically, Convolution is an operation by which a given filter consisting of certain weights applies to the input image, then convolves the image pixel by pixel, performs an operation [29] and the output from the convolution is carried forward for further processing. CNN's are comprised of an input layer, a group of hidden layers and an output layer. Usually, hidden layers in CNN include convolutional layers, pooling layers, activation layer, fully connected layers, and normalization layers. The output from one layer is fed as input to the next layer accordingly and finally, the output layer determines the class of the object. The flow chart of CNN architecture is shown in figure 2.1 [14].

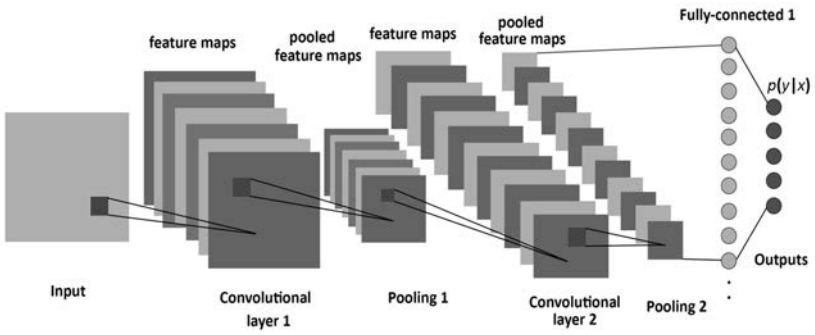

Figure 2.1.: CNN architecture [3] [14]

2.1.1. Input Layer

An image has a certain number of pixels and channels whereas each pixel holds a value of the color. A machine can read images only by the color values or pixel values in different channels. The input layer holds the pixel values of the image. A grayscale image with its pixel values is shown in figure 2.2. A small block of a picture is shown in a table where the value 0 represents black color and 255 represents white color. The other values in between 0 and 255

represent other color levels of grayscale. This is how our local machine splits the image into a table or matrix of pixels [1].

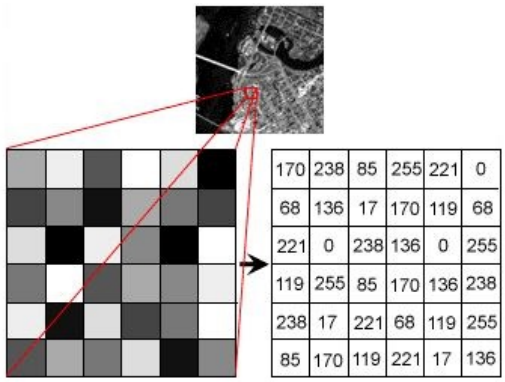

Figure 2.2.: Distribution of pixels in image shown on a large scale to represent pixel values [8].

2.1.2. Convolutional Layer

A Convolutional layer consists of a certain number of filters or neurons. Each filter is designed with its weights in such a way so that after the convolution a specific feature can be extracted [1]. By each iteration of training, the weights are learned such that feature that can be precisely extracted and the losses [2.2] are minimized. If one filter determines the presence of edges in images, other filter extracts the color value or calculates if there is any corner and so on [1]. In the convolutional layer, the input image is convolved and the dot products output after the convolution with each shift of one pixel by one pixel to left and downwards forms a new matrix [figure 2.3].

2.1.3. Stride and Padding

In Machine Learning, hyperparameters are some tuning parameters which are usually tuned before starting the training to control the behavior of the Machine Learning model [13]. In figure 2.3, it is shown that the filter or matrix is shifted left and downward by one pixel. The filter can be shifted by pixels of predefined number value as well, which can be set as a hyperparameter. In CNN architecture, this hyperparameter is called Stride. If stride is set to 2, then the matrix will move by 2 pixels at a time [figure 2.4].

Padding means adding a zero or non-zero column or row at the end or beginning of a matrix. The main purpose of padding is to prevent data at the edges

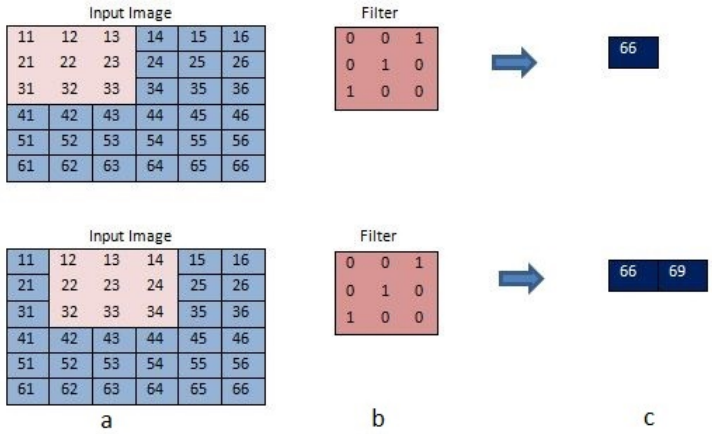

Figure 2.3.: Initial steps of convolution, a 3×3 filter (b) applied to an image matrix (a) results in (c)

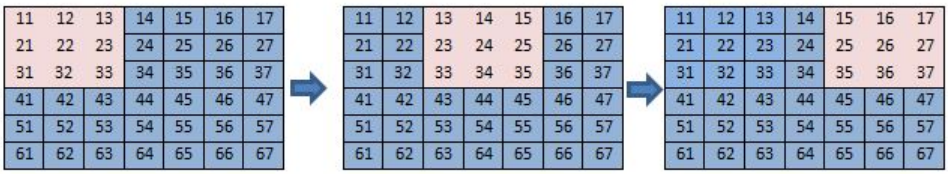

Figure 2.4.: Convolution with Stride=2, filter (red color) moves by 2 pixels after each step

of the image from being lost after convolution. Padding with zero columns and rows is known as zero-padding [figure 2.5]. As the resulting image shape decreases with the increase of stride value, padding helps it to keep balance on image shape [1]. Padding can also be done by adding two zero columns in case if the stride value is two.

2.1.4. Pooling Layer

Sometimes it happens that the size of the image is extremely large and it needs to be reduced. The pooling layer (also known as the downsampling layer) applies a filter of a given size (e.g. 2×2) to an image that combines the pixel values into one. There are two different types of pooling normally used in CNN; max pooling and average pooling. Max pooling takes the maximum value from the cluster, whereas the average pooling calculates the average value from the

0	0	0	0	0	0	0	0
0	11	12	13	14	15	16	0
0	21	22	23	24	25	26	0
0	31	32	33	34	35	36	0
0	41	42	43	44	45	46	0
0	51	52	53	54	55	56	0
0	61	62	63	64	65	66	0
0	0	0	0	0	0	0	0

Figure 2.5.: Zero padding; zero columns (light blue) are added at the edge of the image matrix (blue)

cluster where the pooling filter is applied figure 2.6.

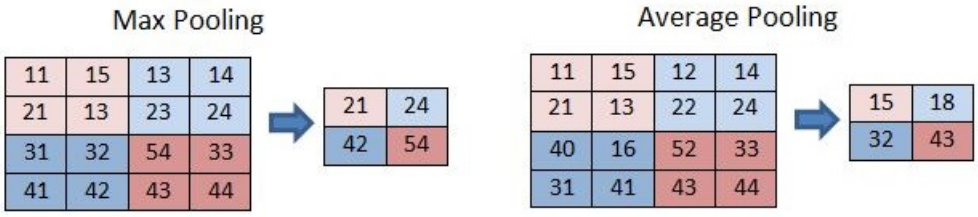

Figure 2.6.: Max Pooling and Average Pooling

2.1.5. Output Dimension

The output image after the convolution is denoted as an activation map. Since after the convolution with each of the filter the size of the output image deviates from its original size, after a few filters applied on the input image, the size of the output image is hard to predict. The three following parameters affect the shape of the output image.

Number of Filters: The depth of the activation map is the same as the number of filters applied on the input image. For example, if ten 3 x 3 filters are applied on an input image, the depth of the activation map will be 10 and the other size parameters will vary according to stride value and zero padding.

Stride: An increase in stride value decreases the number of rows and columns of the activation map accordingly. As in figure 2.4, it is observed that a stride value of 1 has decreased the rows and columns by 2. Similarly, a stride value of 2 will decrease the rows and columns by 4.

Zero Padding: Zero paddings add zero column and zero rows in input image before convolution, which in result increases the dimension in the activation map.

The dimension size of the activation map can be measured by a simple formula: ([W-F+2P]/S) +1, where, F is the size of the filter, W is the input image dimension, P is the number of padding applied and S is the Stride value.

Suppose 10 filters of size 5*5*3 (F=5), with single stride (S=1) and no zero padding (P=0) are applied on a RGB color image of size 32*32*3 (W=32), the dimension of the activation map will be ([(32-5+0)/1] +1=) 28 and the depth will be of the number of filters applied on that image i.e. 10. Therefore, the activation map will have a size of 28*28*10.

2.1.6. Output Layer

After the convolution and padding in the different layers are performed, the output layer then determines the class in which the object belongs to. The output layer will have the same number of neurons as the number of classes [3.1]. As already all of the features are properly detected and sizes of the parameters are sufficiently reduced, the data is then transmitted to the neurons of the output layer. Through the activation function, the weighted sum of the output layer neurons identifies the particular class of this object (figure 2.7)[2]. The output layer also calculates a loss function [2.2] according to model designs.

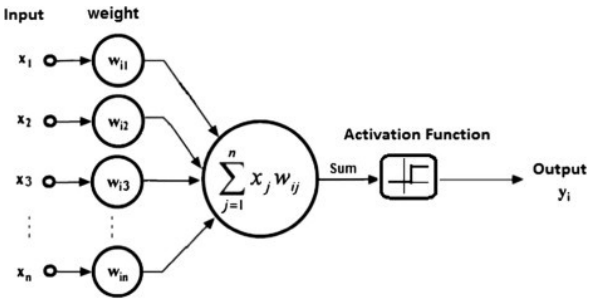

Figure 2.7.: Output layer. The weighted sum of the input features is fed to the Activation function, which determines the class where it belongs to.

Once the forward propagation is complete and the loss does not go under a predefined threshold value, the error is then fed back to the network again and backpropagation begins to update the weight and biases for error and loss reduction.

2.2. Single Shot MultiBox Detector (SSD)

SSD model was first published at the end of November 2016. The main idea behind designing SSD is to overcome the drawbacks of Faster R-CNN regarding the training speed issue. Faster R-CNN is a region-based Convolutional Neural Network. Faster R-CNN consists of 2 different networks: region proposal network (RPN) for generating region proposals and a network using these proposals to detect objects [5] [31]. In SSD, object localization and classification are processed by a feed-forward pass of the designed convolutional network, which produces a collection of rectangle boxes with different fixed-size shapes and scores that represents the probability of the presence of objects [27]. Non-maximum suppression is then applied to find out the final detection among all of the boxes. SSD is comprised of 22 different layers. Structurally, Visual Geometry Group-16 (VGG-16) network is used as a base network [figure 2.8]. VGG is a convolutional neural network used for feature extraction and VGG-16 refers to the VGG model along with 16 weight layers. SSD adds 6 more auxiliary convolution layers after the VGG-16 [11].

A group of convolutional feature layers with a gradual decrease in size are added at the end of the base network to obtain the predictions of detections at multiple scales. Each added feature layer produces a fixed set of detection predictions applying a set of convolutional filters. In order to form a feature map

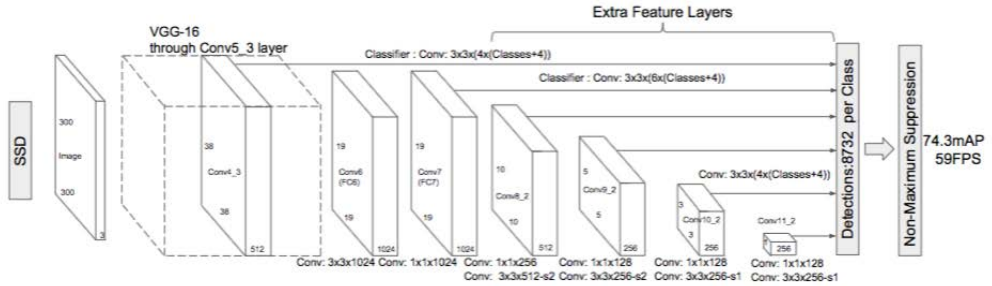

Figure 2.8.: SSD model architecture [27]. SSD adds 6 more auxiliary convolution layers after the VGG-16 [11].

of size m x n with p channels, the size of the basic element which eventually predicts a potential parameter and provides either a score or a rectangle box shape related to the default box parameter is of 3 x 3 x p sized small kernel.

Losses: Once the detection or rectangle boxes are predicted, the next step is to calculate the amount of losses. During prediction, the SSD model calculates two basic losses.

- Localization Loss
- Confidence Loss

Localization loss refers to how much the predicted rectangle boxes are shifted or deviated from the ground truth rectangle box. Localization loss is expressed mathematically by equation (2.2.1). x_{ij}^{k} is an indicator for matching the i^{th} default box to the j^{th} ground truth box of k class. N is the total number of default boxes. l is the parameter of the default bounding box and g is the parameter of the ground truth rectangle box. The parameters are the center of the rectangle (cx, cy), width (w) and height (h). Localization loss between the predicted box l and the ground truth box g is defined as the smooth L1 loss.

$$L_{loc}(x,l,g) = \sum_{i \in Pos}^{N} \sum_{m \in cx,cy,w,h} X_{ij}^{k} smooth_{L1}(l_i^m - \hat{g}_j^m) \tag{2.2.1}$$

$$\hat{g}_j^c x = (g_j^{cx} - d_i^{cx})/d_i^w \tag{2.2.2}$$

$$\hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h \tag{2.2.3}$$

$$\hat{g}_j^w = \log(\frac{g_j^w}{d_i^w}) \tag{2.2.4}$$

$$\hat{g}_j^h = \log(\frac{g_j^h}{d_i^h}) \tag{2.2.5}$$

Confidence loss is mainly the classification loss calculated by equation (2.2.6). It is obtained from the Softmax function from the Tensorflow's Python packages. The Softmax function measures the probability that the training sample belongs to a particular class as compared to belongs to other classes [equation (2.2.7)].

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in Neg} \log(\hat{c}_i^o) \tag{2.2.6}$$

where

$$\hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)} \tag{2.2.7}$$

Total loss is the weighted sum of localization loss and classification loss [equation (2.2.8)].

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g) \tag{2.2.8}$$

3. Training Procedure

The main idea behind the study is to detect air bubbles and brine channels in Arctic sea ice. Therefore, the objects are air bubbles and brine channels for what a convolutional neural network (CNN) is trained with a good number of training samples. Training samples are divided into 2 different datasets. One is a training data set and another one is test data set. The train data set is used to train the model or to be processed to extract the significant features and the test data set is used for evaluating the features and calculating the losses where the extracted features are applied for detection.

3.1. Terms Related to Training Step

Some important terms which are related to training steps are:

Class: In Convolutional Neural Network, class refers to the number of different possible objects, for what the network is trained. For example, our CNN model is trained for 2 different kinds of objects (i.e., air bubble and brine channel) detection. So the number of classes is 2.

Weight: Each of the convolutional layers has multiple neurons or filters. Each of the values of filters is known as weight. When a filter is applied on an input image matrix, the elements of the input matrix are multiplied with the weights of the filter. A weight with zero value means that the feature is not significant.

Bias: In addition to the weights, there is another linear component which is added with the result of multiplication between weight matrix and input image matrix. The linear component element is called Bias. For example, if the input image matrix is a and a weight element of the filter is $W1$, then the operation of weights and bias b can be expressed as $[(a * W1) + b]$

Activation Function: After multiplication by weights and addition by bias, the activation function is applied to the result to transform the input

signal to the output signal in a layer. The most common activation functions used in CNN are Sigmoid function, Rectified Linear Unit (ReLU) and Softmax [35].

Forward Propagation: The output signal of the activation function of a layer is then fed to the next layer. The movement of input through the convolutional layers to the output layer is called forward propagation.

Gradient Descent: If the extracted feature does not predict the object well, it results in large amount of loss. In order to optimize the loss function, the value weights and biases are needed to increase or decrease slightly so that minimum loss point is met.

Learning rate: The amount of increase or decrease in weights and biases is known as the learning rate. A large value in learning rate returns large fluctuation in total loss and small value in learning rate controls the curve smoothly.

Back Propagation: The update in weights and bias is then fed again to the first convolutional layer so that the CNN model extracts more significant features than before so that the loss is minimized. This process of feeding back the values is called Back Propagation.

Iteration: Iteration is a single time procession of the forward propagation, gradient descent, and the back propagation. That means when each of these processes is done for once and all of them are completed, it is then said a single iteration is finished.

Batch Size: It is a parameter in machine learning by which the number of training samples is mentioned which will be utilized on a single iteration for feature extraction. Depending on the speed of the system where the training will be processed, the batch size has to be chosen.

Epoch: Once all of the training samples are processed of forward and back propagation for once, then it is considered as 1 epoch.

For example, the total number of training samples is 100, the batch size is defined as 10, therefore, once 10 samples are processed it is measured as 1 iteration. So the number of iterations needed for 1 epoch is (total sample

divided by the batch size i.e. 100/10 =) 10. After the training is complete, the total number of the epoch can be calculated by dividing the number of iterations by the number of iterations needed for 1 epoch (i.e. 10).

3.2. Training Steps

Ice cores are the cylindrical ice samples obtained across the entire depth of an ice floe in the Arctic sea by employing ice core auger [19]. Ice cores are cut into the height of 10 centimeters slices by using chop. Then using a microtome, rectangular shaped cross-sections with a thickness of 2 meters are obtained from the slices which are called thick sections of the ice core. After that, images are captured from the thick sections.

A total of 78 thick sections images from 12 different ice cores are provided for training and testing the CNN model. The model is applied to the thick section images of 8 different ice cores. For the demonstration purpose, air bubble and brine channel detection images and plots of area, circularity and Provability Density Function (PDF) for 5 thick section images of only the Core 1 [A.4], which was provided by N-ICE2015 Campaign [22][32] are shown here. In chapter 5, how the number of air bubbles is changing with respect to the depth of the ice core is summarized for 5 different ice cores. Typical thick section images for Core 1 [A.4] is shown in figure 3.1. The air bubble and brine channel, which are detected, are shown in figure 3.2 and figure 3.3. Some typical air bubbles occurrence in the ice core are shown in figure 3.4 with blue rectangles.

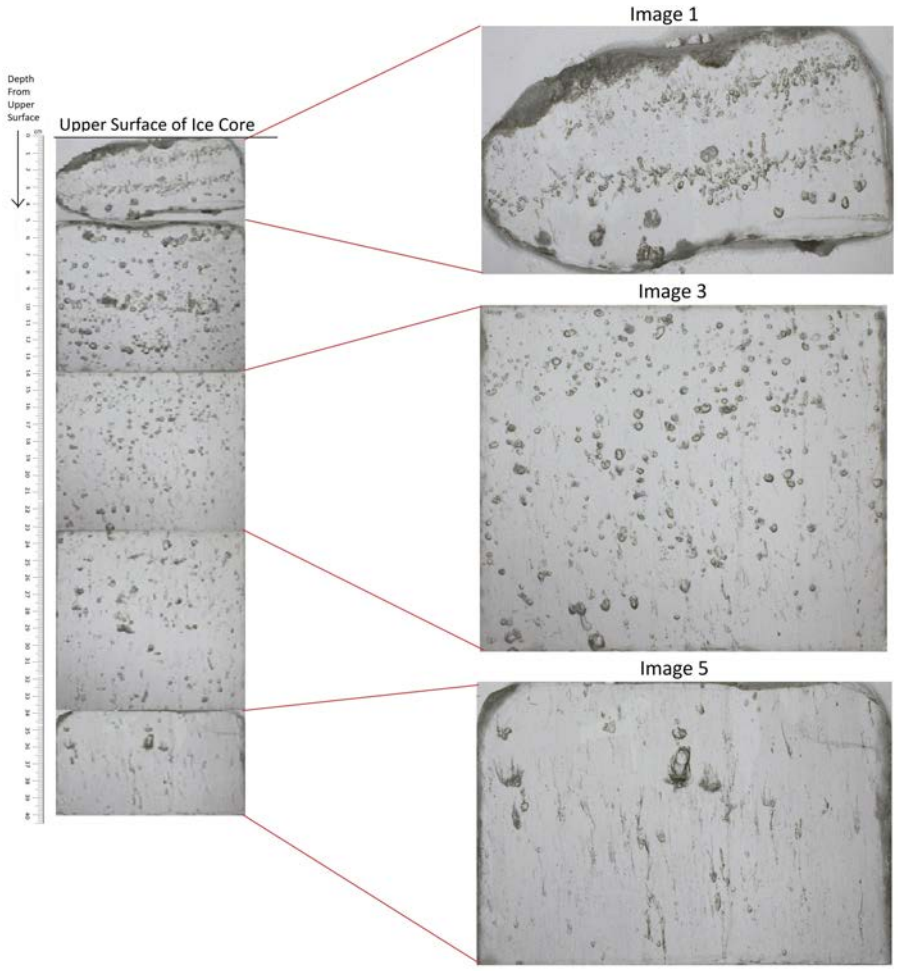

Figure 3.1.: Ice core images arranged from top of the core(provided by N-ICE 2015 campaign [22]. Centimeter scale is shown at the left. The 1st, 3rd and 5th image of the ice core are shown on large scale.

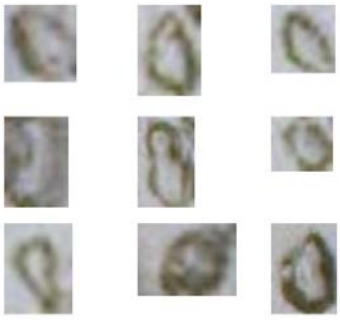

Figure 3.2.: Some typical air bubble samples

Figure 3.3.: Some typical brine channel samples

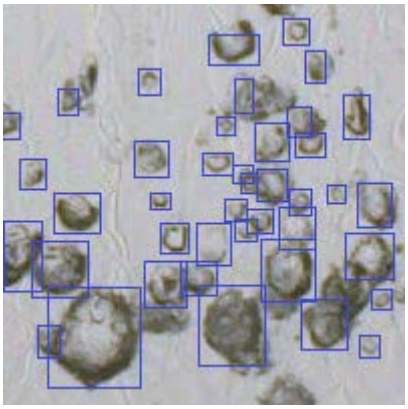

Figure 3.4.: Some typical air bubbles in the Arctic sea ice (blue rectangle)

Annotation is a term used in machine learning refers to label the objects manually by drawing a rectangle box. By annotation, the information about the location and dimension of the object can be delivered to the CNN model. As the shape of each image was extremely large (average size is approximately 1800×1800 pixels), all of the JPEG images are cropped into 200×200 pixels sample PNG format image in order to release the complexity of Annotation. After conversion to 200×200 pixels, the total amount of sample images is now 7128, where all of the samples don't contain the desired object itself and even a good number of samples contain core edge or noise. Therefore, a sum of 1397 samples of 200×200 pixels size is manually filtered from 7126 samples for annotation. A few researchers suggest to divide the whole dataset into a training dataset and test dataset following an approximately 3:1 ratio [12],[10]. Keeping in mind that in order to obtain a potential detector, the number of training samples has to be more than the number of test samples, amongst 1397 samples, 939 samples are annotated as train samples and the rest 458 sample is annotated as test samples using the annotation tool *labelImg* [A.2]. After annotation in a single image, an XML file is automatically generated by the *labelImg* which explains the location and dimensions of the annotated object. Once the annotation is completed, all of the generated .xml files from the training sample and test sample are converted to a separate single CSV file, which is delivered to the CNN model for training.

Before starting training, one important step is to set the hyperparameters. Therefore some important hyperparameters in the configuration file are set in the following way and the other hyperparameters are kept unchanged from the default value. All configuration information are written in Appendix section

A.3.

```
num_classes: 2
image_resizer {
    fixed_shape_resizer {
      height: 200
      width: 200
    }
  }
train_config: {
batch_size: 12
optimizer {
  rms_prop_optimizer: {
    learning_rate: {
      exponential_decay_learning_rate {
        initial_learning_rate: 0.0002
        decay_steps: 800720
        decay_factor: 0.95
      }
    }
    momentum_optimizer_value: 0.9
    decay: 0.9
    epsilon: 1.0
  }
}
```

Once all hyperparameters are set, then the next step is training the SSD model for the dataset. SSD model starts with a loss of about 20, and it is recommended to train the model until the loss is consistently under 2. When the training is started, the loss is started decreasing from a value 20.3640. The beginning steps of training are shown in figure 3.5.

The model is trained up to 300000 steps of iteration. The model is trained on the system called *exzellnc021*. The hardware information of *exzellnc021* is briefly written in Appendix A.5. The total training time required on that system is around 56 hours. Training steps at the end with the amount of loss can be visualized in figure 3.6. It shows the amount of total loss is reached to around 2.3

The model is trained up to 300000 steps of iteration. It took around 56 hours of time to reach this level. Training steps at the end with the amount of loss

```
INFO:tensorflow:Recording summary at step 0.
INFO:tensorflow:global step 1: loss = 18.5128 (13.755 sec/step)
INFO:tensorflow:global step 2: loss = 19.8451 (0.701 sec/step)
INFO:tensorflow:global step 3: loss = 20.3640 (0.666 sec/step)
INFO:tensorflow:global step 4: loss = 18.9056 (0.660 sec/step)
INFO:tensorflow:global step 5: loss = 17.6361 (0.619 sec/step)
INFO:tensorflow:global step 6: loss = 18.7532 (0.638 sec/step)
INFO:tensorflow:global step 7: loss = 20.2091 (0.636 sec/step)
INFO:tensorflow:global step 8: loss = 16.6177 (0.640 sec/step)
INFO:tensorflow:global step 9: loss = 17.4534 (0.634 sec/step)
INFO:tensorflow:global step 10: loss = 17.2100 (0.663 sec/step)
INFO:tensorflow:global step 11: loss = 16.2401 (0.638 sec/step)
INFO:tensorflow:global step 12: loss = 17.5278 (0.643 sec/step)
INFO:tensorflow:global step 13: loss = 16.5059 (0.629 sec/step)
INFO:tensorflow:global step 14: loss = 16.2284 (0.632 sec/step)
INFO:tensorflow:global step 15: loss = 16.7602 (0.620 sec/step)
INFO:tensorflow:global step 16: loss = 15.2676 (0.634 sec/step)
INFO:tensorflow:global step 17: loss = 16.1222 (0.662 sec/step)
INFO:tensorflow:global step 18: loss = 15.0429 (0.678 sec/step)
INFO:tensorflow:global step 19: loss = 15.1376 (0.665 sec/step)
INFO:tensorflow:global step 20: loss = 16.1036 (0.688 sec/step)
INFO:tensorflow:global step 21: loss = 14.8522 (0.687 sec/step)
INFO:tensorflow:global step 22: loss = 14.9730 (0.648 sec/step)
INFO:tensorflow:global step 23: loss = 14.7718 (0.636 sec/step)
INFO:tensorflow:global step 24: loss = 14.6822 (0.633 sec/step)
INFO:tensorflow:global step 25: loss = 14.9580 (0.631 sec/step)
INFO:tensorflow:global step 26: loss = 13.5814 (0.619 sec/step)
```

Figure 3.5.: Training steps at the beginning of training. Calculated loss (at step 1=18.5128) is gradually decreasing (at step 26=13.5814).

can be visualized in figure 3.6. It shows the amount of total loss is reached to around 2.3.

```
INFO:tensorflow:global step 299975: loss = 3.6131 (0.679 sec/step)
INFO:tensorflow:global step 299976: loss = 3.5011 (0.676 sec/step)
INFO:tensorflow:global step 299977: loss = 3.8310 (0.677 sec/step)
INFO:tensorflow:global step 299978: loss = 2.5846 (0.693 sec/step)
INFO:tensorflow:global step 299979: loss = 2.2745 (0.717 sec/step)
INFO:tensorflow:global step 299980: loss = 3.4492 (0.702 sec/step)
INFO:tensorflow:global step 299981: loss = 2.7088 (0.725 sec/step)
INFO:tensorflow:global step 299982: loss = 2.6729 (0.695 sec/step)
INFO:tensorflow:global step 299983: loss = 3.6341 (0.652 sec/step)
INFO:tensorflow:global step 299984: loss = 3.3977 (0.667 sec/step)
INFO:tensorflow:global step 299985: loss = 2.1499 (0.685 sec/step)
INFO:tensorflow:global step 299986: loss = 3.2886 (0.667 sec/step)
INFO:tensorflow:global step 299987: loss = 4.2621 (0.709 sec/step)
INFO:tensorflow:global step 299988: loss = 1.8929 (0.674 sec/step)
INFO:tensorflow:global step 299989: loss = 2.8598 (0.694 sec/step)
INFO:tensorflow:global step 299990: loss = 3.3516 (0.684 sec/step)
INFO:tensorflow:global step 299991: loss = 1.9271 (0.703 sec/step)
INFO:tensorflow:global step 299992: loss = 2.0605 (0.711 sec/step)
INFO:tensorflow:global step 299993: loss = 3.3005 (0.696 sec/step)
INFO:tensorflow:global step 299994: loss = 3.0886 (0.695 sec/step)
INFO:tensorflow:global step 299995: loss = 2.0592 (0.682 sec/step)
INFO:tensorflow:global step 299996: loss = 2.8550 (0.697 sec/step)
INFO:tensorflow:global step 299997: loss = 3.0394 (0.689 sec/step)
INFO:tensorflow:global step 299998: loss = 2.3891 (0.683 sec/step)
INFO:tensorflow:global step 299999: loss = 2.6675 (0.673 sec/step)
INFO:tensorflow:global step 300000: loss = 2.3890 (0.692 sec/step)
INFO:tensorflow:Stopping Training.
INFO:tensorflow:Finished training! Saving model to disk.
```

Figure 3.6.: Training steps at the end of training. Calculated loss is 2.3890 at final step (at step 300000).

Therefore the number of Epoch can be calculated from the total number of training iteration divided by the number of iterations needed for processing all of the samples for once. The number of iterations needed for processing all of the samples for once can be obtained from the total number of training samples divided by the batch size. That is (i.e. 939/12 =) 78.25. Therefore, the number of Epoch is (300000/78.25 =) 3834.

Tensorboard is a visualization tool of Tensorflow. While the model is training, localization loss, classification loss, and total loss is observed from Tensorboard. During the training, Tensorboard can be opened from any browser by running the following command in command terminal.

```
tensorboard --logdir=training
```

After training has completed, all of the loss graphs are collected from the Tensorbord of the system server. Classification loss, localization loss and total loss graph are shown in figure 3.7, figure 3.8, figure 3.9.

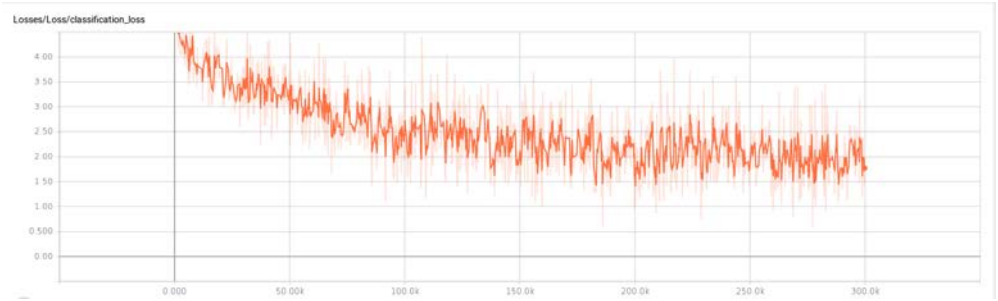

Figure 3.7.: Classification loss. The curve (orange color) represents the continuous reduction of classification loss over the training steps.

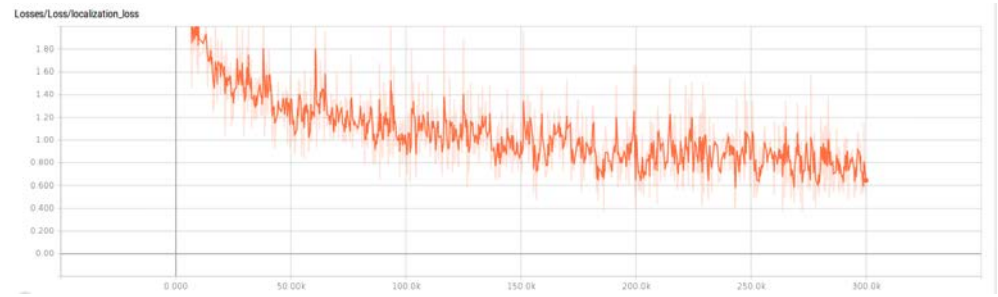

Figure 3.8.: Localization loss. The curve (orange color) represents the continuous reduction of localization loss over the training steps.

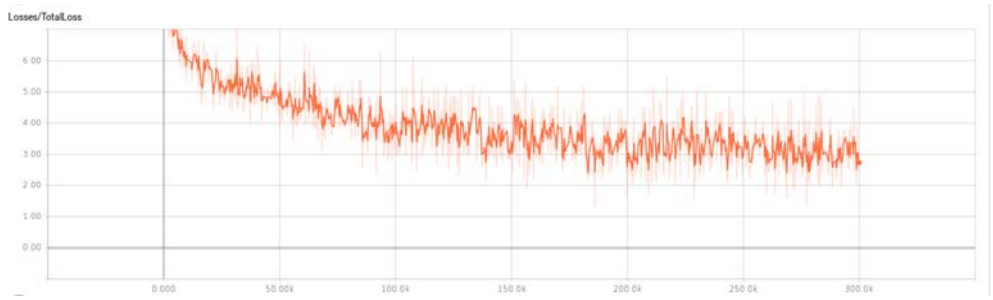

Figure 3.9.: Total loss curve; sum of the classification loss and localization loss

3.3. Bubble Detection Results on Image

A checkpoint is a term related to CNN model training, referred to as the training records for a certain breakpoint. By default, the checkpoint is set to be saved after every 223 steps and the number of the latest checkpoint to be saved is set to 5. In our training, the number of the latest checkpoint to be saved is changed to 30, so that a variety of checkpoints can be evaluated to obtain our best detection result.

After that, the latest checkpoint which is saved at step number 299205, is used to generate our *frozen_ inference_ graph*. The *frozen_ inference_ graph* is a record file of all features extracted during training steps that work as an object classifier. So the *frozen_ inference_ graph* is our main air bubble and brine channel detector and can be applied to the image directly. The trained classifier is applied to 5 top images of a multi-year ice core 1 [A.4]. The images of Core 1 are independent and not a part of the train and test image set. The images before and after classifier applying are shown below (from figure 3.10 to figure 3.19). On the resulting images where the detector is applied, the green rectangles represent air bubbles and yellow rectangles represent brine channels. The thick section images of multi-year ice core 1 [A.4] are arranged in such a way that Image 1 is taken from top of the core and the objects of Image 1 are located very near to upper sea surface and Image 5 is taken from bottom of the core and the objects of Image 5 are located in depth of around 40 centimeters from the top.

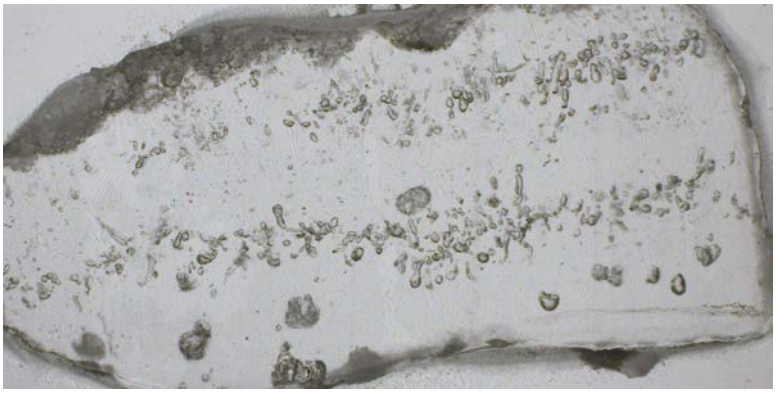

Figure 3.10.: 1st section from top of multiyear ice core 1 obtained from N-ICE 2015 Campaign (A.4) before air bubble and channel detection; width:10cm; height:5cm

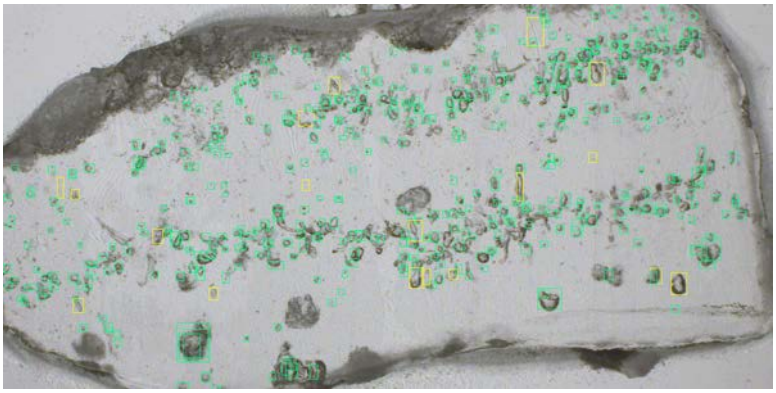

Figure 3.11.: 1st section from top of multiyear ice core 1 obtained from N-ICE 2015 Campaign (A.4) after air bubble and channel detection; width:10cm; height:5cm; analysed on 18.06.2019)

Figure 3.12.: 2nd section from top of multiyear ice core 1 obtained from N-ICE 2015 Campaign (A.4) before air bubble and channel detection; width:10cm; height:9cm

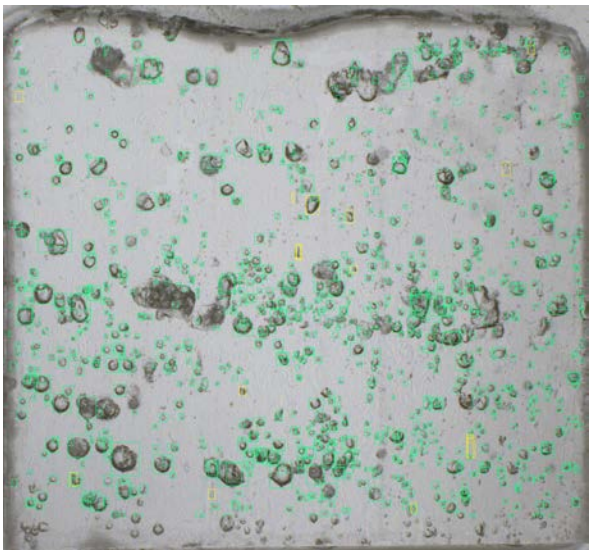

Figure 3.13.: 2nd section from top of multiyear ice core 1 obtained from N-ICE 2015 Campaign (A.4) after air bubble and channel detection; width:10cm; height:9cm

Figure 3.14.: 3rd section from top of multiyear ice core 1 obtained from N-ICE2015 Campaign (A.4) before air bubble and channel detection; width:10cm; height:9cm

Figure 3.15.: 3rd section from top of multiyear ice core 1 obtained from N-ICE2015 Campaign (A.4) after air bubble and channel detection; width:10cm; height:9cm

Figure 3.16.: 4th section from top of multiyear ice core 1 obtained from N-ICE2015 Campaign (A.4) before air bubble and channel detection; width:10cm; height:11cm

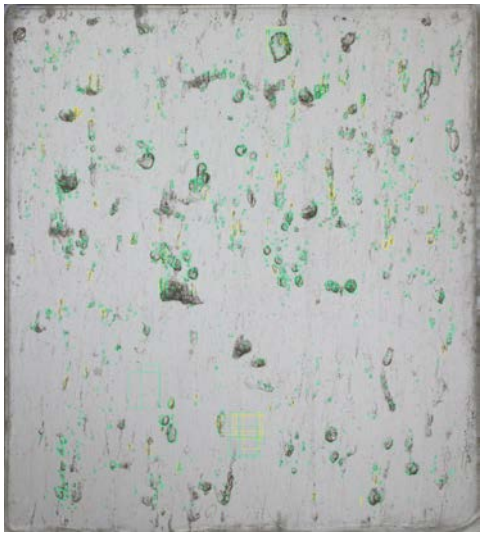

Figure 3.17.: 4th section from top of multiyear ice core 1 obtained from N-ICE2015 Campaign (A.4) after bubble and channel detection; width:10cm; height:11cm

Figure 3.18.: 5th section from top of multiyear ice core 1 obtained from N-ICE2015 Campaign (A.4) before air bubble and channel detection; width:10cm; height:6cm

Figure 3.19.: 5th section from top of multiyear ice core 1 obtained from N-ICE2015 Campaign (A.4) after air bubble and channel detection; width:10cm; height:6cm

3.4. Performance Evaluation

The model is able to detect air bubbles accurately in most cases. Brine channels, however, are detected less reliably and are missed in many cases. As the model's brine channel detection is still unsatisfactory, the performance of the brine channel detection is not evaluated. The following terms will be used to evaluate the performance of the air bubble detection.

True Positive(TP) are the number of correctly identified air bubble as air bubble by detector.

False Positive(FP) are the number of objects which are not air bubble, but incorrectly identified as air bubble.

Precision Rate is the ratio of True Positive (TP) to the sum of True Positive (TP) and False Positive (FP) [18].

True Positive and False Positive are manually counted.

Test Sample	True Positive	False Positive	Precision Rate
Image 1	451	37	92.42%
Image 2	934	66	93.4%
Image 3	772	44	94.61%
Image 4	792	225	77.87%
Image 5	322	113	74.03%

Table 3.1.: Result of air bubble detector

4. Analysis of Statistical Information of Air Bubble

4.1. Extraction of Air Bubble Contours

It is already clear to us that the above-explained model detects rectangles covering the air bubble, rather than solely the air bubble. After the air bubbles in the rectangle are properly detected, the next task is to extract a pure air bubble from the rectangle and then to derive the area, perimeter, and circularity of the air bubble. In order to extract the pure bubble, contours approximation method is applied on the rectangle. Contour is a curve or a boundary drawn by joining all continuous points of same colour or intensity. OpenCV libraries have functionality to find the contours in an image, which also provides the area of the contours and circumference of the contours [4].

As the contour approximation method works more accurately on the binary image, binary thresholding is applied, at first, to the air bubble sample image. As binary thresholding works only on a grayscale image, the color space of the air bubble sample image is converted to grayscale color space. Therefore, if the color space of the sample image is grayscale, it remains unchanged, otherwise, it converts to grayscale color image. For the binary thresholding, the average pixel value is set as a binary threshold value. The average pixel value is the ratio of the sum of all pixel values to the number of the total pixels (i.e. rows multiplied by columns). The result of binary thresholding on an air bubble sample image is shown in figure 4.1.

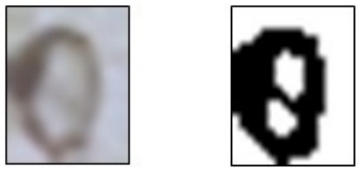

Figure 4.1.: Air bubble sample image (left) and binary thresholded image (right), Threshold value (T) = mean pixel value

The contour approximation method provided by OpenCV extracts different brighter pixel regions separated by the darker pixel boundaries. As a bubble is a region of brighter pixels surrounded by a comparatively darker boundary, it may provide more than one region, which needs another filtering process to distinguish one region from another to obtain the air bubble area. Contour approximation is applied to the air bubble sample image of figure 4.2, which results in 4 regions [figure 4.1]. Our Region of Interest (ROI) is the black region of the image including the inner 2 white regions, which is more complex to extract.

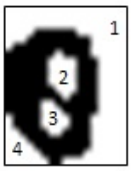

Figure 4.2.: Contour approximation applied on binary thresholded image, detected contours are marked by 1,2,3 and 4

To overcome this problem, the binary inversion is applied to the binary thresholded image. Binary inversion refers to convert the white pixels to black pixels and black pixels to white pixels. After that, contour approximation is applied to the binary inverted bubble image, which results in only one region. OpenCV has functionality which checks for convexity or cavity inside a Region of Interest (ROI) and corrects it [figure 4.3].

Figure 4.3.: Binary thresholded image (left), binary inverted image (middle)(detected contour marked by 1), cavity filled image (right)

A few air bubbles with contours detection results are shown in figure 4.4. In

the case, if a rectangle contains only a part of the bubble or even doesn't contain an air bubble itself, detection of bubble contour by contour approximation method would be ambiguous. Such kind of partial bubble contour detection shown in figure 4.5. In figure 4.5, it is observed that there is more than one air bubble in the rectangle obtained from the air bubble detection model. As three bubbles are located very nearly from one another, contour approximation method detected one region covering these three bubbles, that's why calculated area and perimeter for the whole region as one bubble.

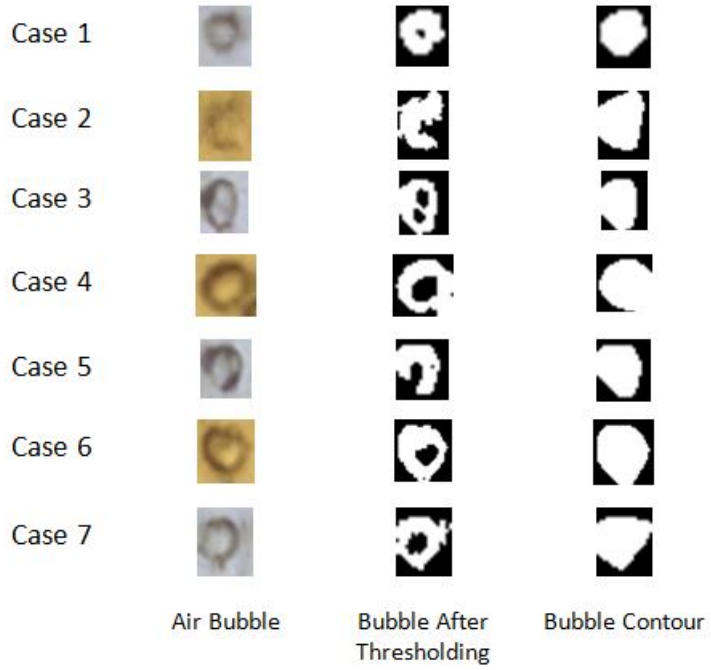

Figure 4.4.: Contour approximation applied on air bubble rectangle

Figure 4.5.: Contour approximation applied on partial-bubble rectangle

Statistical information like area and perimeter of the contour are obtained

from the contour, which are used for further analysis. The area and perimeter calculated for the air bubbles shown in figure 4.4, is narrated in table 4.1

Test Sample	Rectangle Area	Bubble Contour Area	Perimeter
Case 1	195 Pixels	74 Pixels	33.55
Case 2	660 Pixels	268 Pixels	103.74
Case 3	374 Pixels	175 Pixels	51.79
Case 4	506 Pixels	300 Pixels	72.53
Case 5	378 Pixels	135 Pixels	66.63
Case 6	990 Pixels	517 Pixels	93.25
Case 7	396 Pixels	174 Pixels	63.94

Table 4.1.: Rectangle area, air bubble contour area, perimeter of 7 different cases shown in figure 4.4

4.2. Analysis of Shape and Statistical Properties of Air Bubble

All of the core images have sizes on the millimeter scale. The area of a single pixel in the millimeter square unit for each image is calculated by dividing the area of the image in millimeter square (i.e. width multiplied by height) by the number of pixels of the image (i.e. rows and columns of image).

Circularity : Circularity is a measure of the roundness of an object. It represents, how close an object is from a true circle or how much an object has deviated from a true circular shaped object with the same area. Circularity is measured by the following formula [6].

$$Circularity = (4 * \pi * Area)/(Perimeter)^2$$

In this point, area and circularity extraction and calculation methods are included in the air bubble detection model. Therefore, once the air-bubble detection model is applied to an image, besides the air bubble detection, it provides Area and Circularity for all bubbles for each of the images in Centimetre scale which are plotted in the graph.

In chapter 3, it is shown that how Air bubble detection model detects Air Bubble rectangle for 5 images (figure 3.10 to figure 3.19 and table **??**) in a

sequence of one core. In this chapter, Areas are calculated for all of the air bubbles and is plotted for those 5 images. Alongside, in order to visualize how the number of air bubbles of the area above 0.5 mm^2 is changing with respect to the depth of cores, summary plots regarding bubble area for 5 different ice cores are included (figure 4.18 to figure 4.21). Here, figure 4.6 to figure 4.10 shows the area of the *ith* bubble in the Y-axis and the arbitrary sample of detected bubble i in the x-axis.

Mean area and Standard Deviation are measured as well by the formula [23] given below. Mean area and Standard Deviation are added on the top of the respective figures.

$$Mean(area) = \frac{\sum_{N=1}^{N} area(N)}{N} \tag{4.2.1}$$

$$SD(area) = \sqrt{\frac{\sum_{N=1}^{N} |Mean - area(N)|^2}{N}} \tag{4.2.2}$$

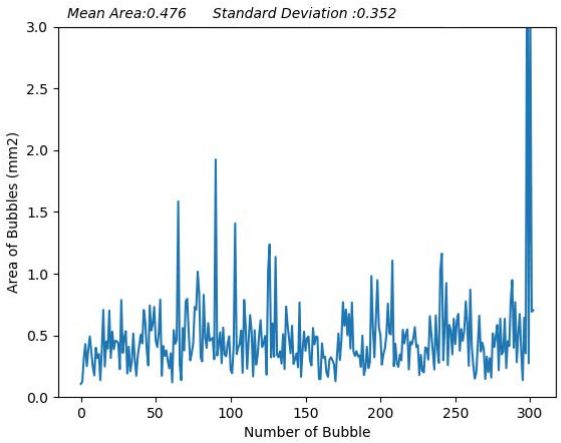

Figure 4.6.: Area of the detected air bubbles in 1st thick section image of ice core 1 [A.4], Depth=0-5 centimetre from the top of the core, x-axis represents the arbitrary sample or detected air bubble i, y-axis represents the area of the corresponding *ith* air bubble.

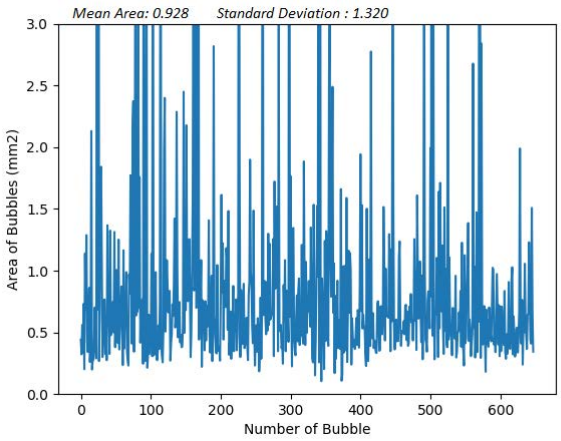

Figure 4.7.: Area of the detected air bubbles in 2nd thick section image of Core 1 [A.4], Depth=5-14 centimetre from the top of the core, x-axis represents the arbitrary sample or detected air bubble i, y-axis represents the area of the corresponding ith air bubble.

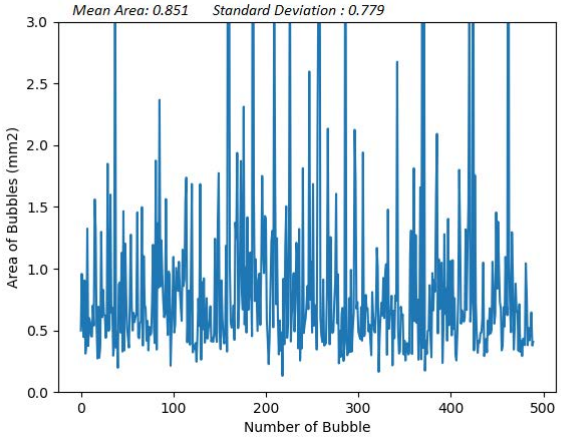

Figure 4.8.: Area of the detected air bubbles in 3rd thick section image of Core 1 [A.4], Depth=14-23 centimetre from the top of the core, x-axis represents the arbitrary sample or detected air bubble i, y-axis represents the area of the corresponding ith air bubble.

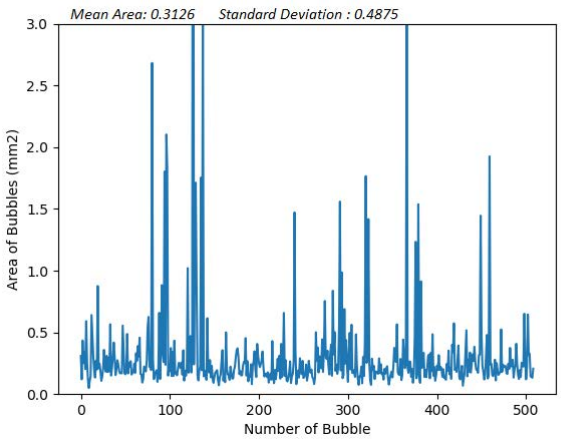

Figure 4.9.: Area of the detected air bubbles in 4th thick section image of Core 1 [A.4], Depth=23-34 centimetre from the top of the core, x-axis represents the arbitrary sample or detected air bubble i, y-axis represents the area of the corresponding ith air bubble.

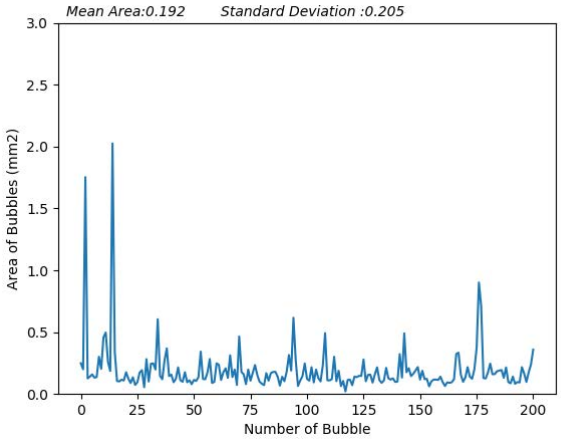

Figure 4.10.: Area of the detected air bubbles in 5th thick section image of Core 1 [A.4], Depth=34-40 centimetre from the top of the core, x-axis represents the arbitrary sample or detected air bubble i, y-axis represents the area of the corresponding ith air bubble.

The circularity of air bubbles in those images is calculated and plotted by the

model. These circularity plots are analyzed by sorting values from minimum to maximum values. Circularity plotted for the 1st thick section of ice core 1 [A.4] is displayed here.(figure 4.11).

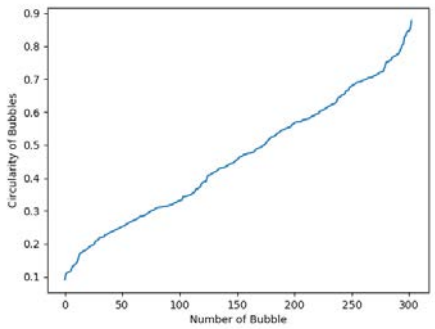

Figure 4.11.: Circularity of detected air bubbles in 1st thick section of ice core 1 [A.4](sorted from minimum to maximum), x-axis represents the arbitrary detected air bubble sample i, y-axis represents the circularity of the corresponding ith air bubble.

From the circularity of air bubbles, Mean Circularity and Standard Deviation (SD) are calculated and summarized in table 4.2.

Core 1	Detected Bubbles	Mean Circularity	Standard Deviation
Image 1	303	0.4597	0.192
Image 2	648	0.4561	0.1884
Image 3	490	0.5061	0.1850
Image 4	509	0.5	0.1789
Image 5	201	0.4580	0.1603

Table 4.2.: Calculated mean and Standard Deviation of circularity for images of Core 1 [A.4]

Probability Density function (PDF): Probability density function (PDF) is simply a curve of continuous random variable points. It represents the relative probability of occurrence of a particular range of discrete values or a particular discrete value [15]. PDF plot is drawn with respect to the area of air bubbles for 5 thick section images of ice core 1(A.4) images (figure 4.12 to figure 4.16).

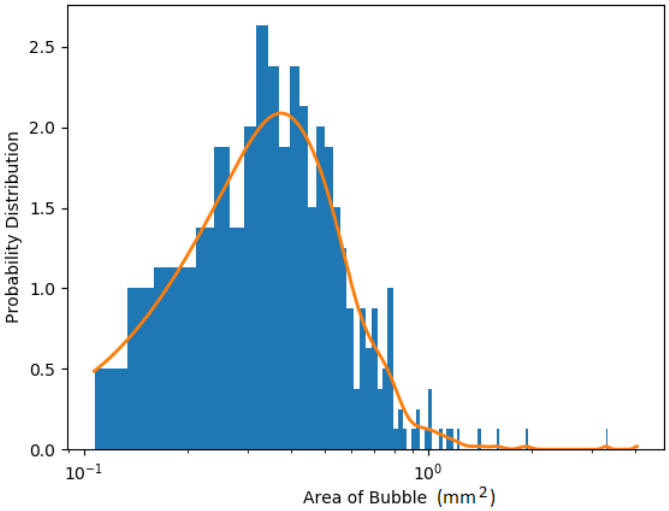

Figure 4.12.: Histogram and PDF with respect to area of air bubbles detected in 1st thick section of sea ice core 1, Histogram drawn in blue color and PDF drawn in orange color, Bin size: 0.0263 [A.4]

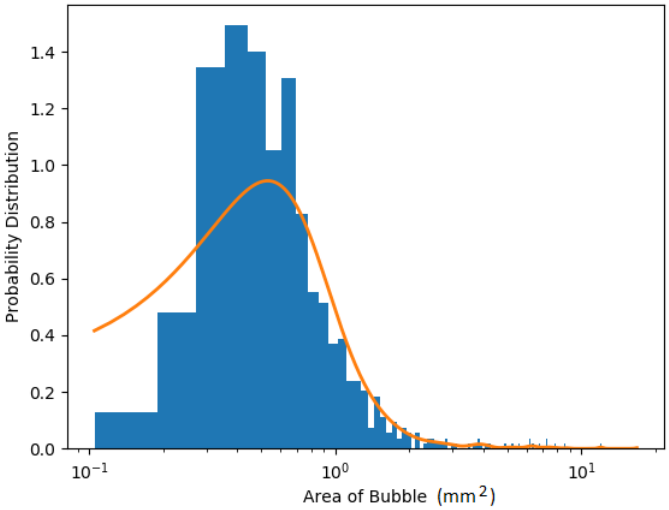

Figure 4.13.: Histogram and PDF with respect to area of air bubbles detected in 2nd thick section of sea ice core 1, Histogram drawn in blue color and PDF drawn in orange color, Bin size: 0.0837 [A.4]

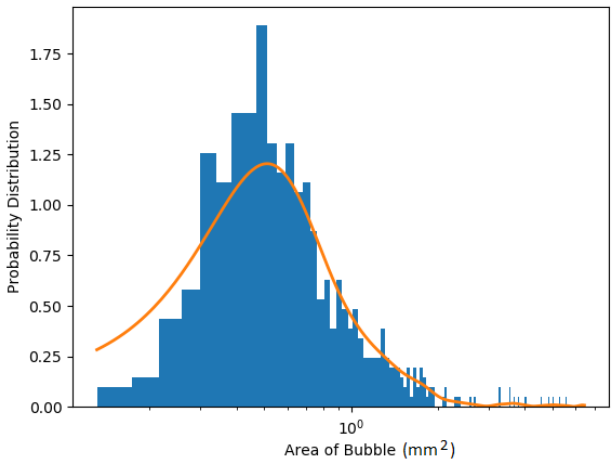

Figure 4.14.: Histogram and PDF with respect to area of air bubbles detected in 3rd thick section of sea ice core 1, Histogram drawn in blue color and PDF drawn in orange color, Bin size: 0.0421 [A.4]

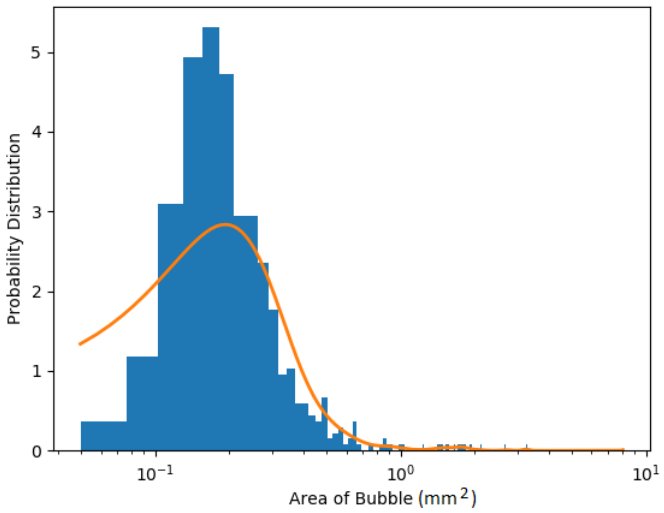

Figure 4.15.: Histogram and PDF with respect to area of air bubbles detected in 4th thick section of sea ice core 1, Histogram drawn in blue color and PDF drawn in orange color, Bin size: 0.0266 [A.4]

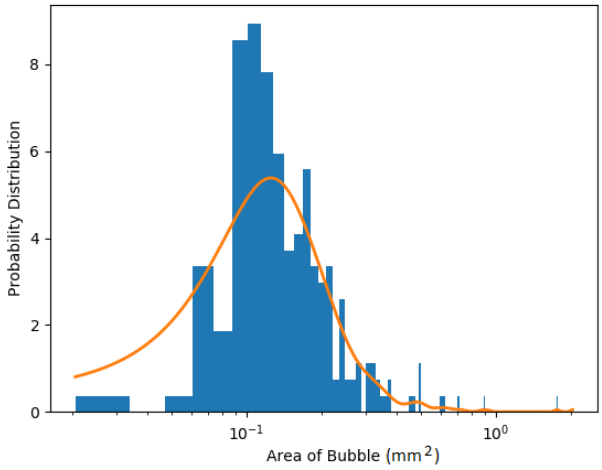

Figure 4.16.: Histogram and PDF with respect to area of air bubbles detected in 5th thick section of sea ice core 1, Histogram drawn in blue color and PDF drawn in orange color, Bin size: 0.0134 [A.4]

For further analysis, air bubbles of area above 0.5 square millimetre is plotted in figure 4.17.

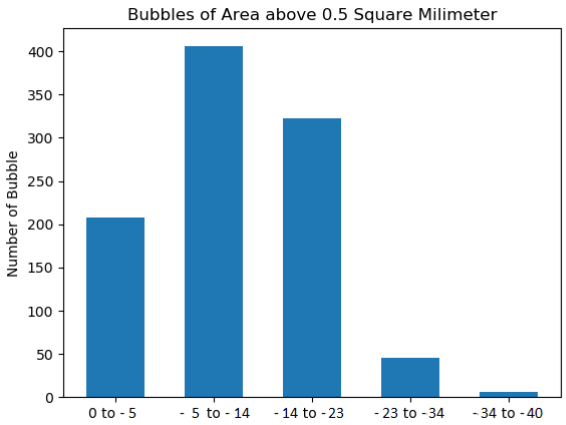

Figure 4.17.: Number of air bubbles with area above 0.5 mm^2 in Core 1 (A.4), x-axis represents the depth of the core in centimeter, y-axis represents the amount of air bubbles found in the mentioned depth of the core

The plot shows that the number of bubbles above a certain area (0.5 mm^2) decreases with the increase of depth. That means the number of big sized bubbles is decreasing as it goes towards the bottom of the sea ice level. As there is not any significant change in the number of bubbles, so the number of smaller bubbles is increasing oppositely. As the heights of Image 1 and Image 5 in Core 1 (A.4) are half (5 Centimetre) of the heights of other Images (10 Centimetre) of Core 1, the number of bubbles for those 2 Images are doubled only for our comparison purpose.

Similarly, air bubbles of the area above 0.5 square millimeters are analyzed for the other cores as well. Here the plots are added for Core 2, Core 3, Core 4, and Core 8 (figure 4.18 to figure 4.21) [A.4].

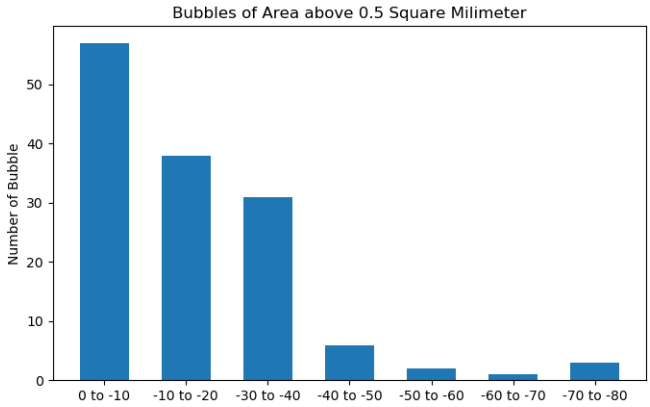

Figure 4.18.: Number of air bubbles with area above 0.5 mm^2 in Core 2 (A.4), x-axis represents the depth of the core in centimeter, y-axis represents the amount of air bubbles found in the mentioned depth of the core

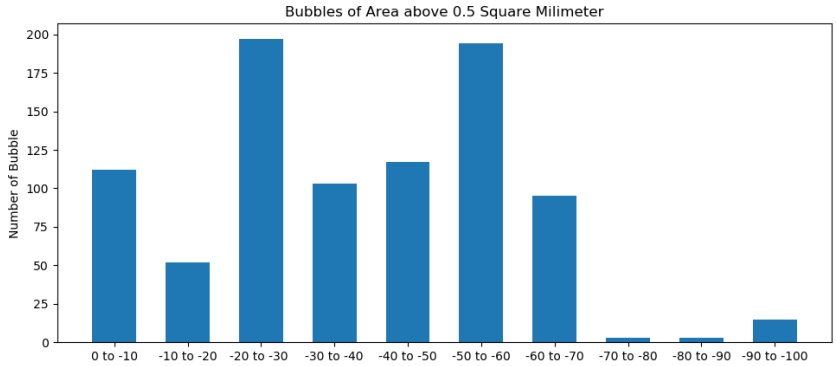

Figure 4.19.: Number of air bubbles with area above 0.5 mm^2 in Core 3 (A.4), x-axis represents the depth of the core in centimeter, y-axis represents the amount of air bubbles found in the mentioned depth of the core

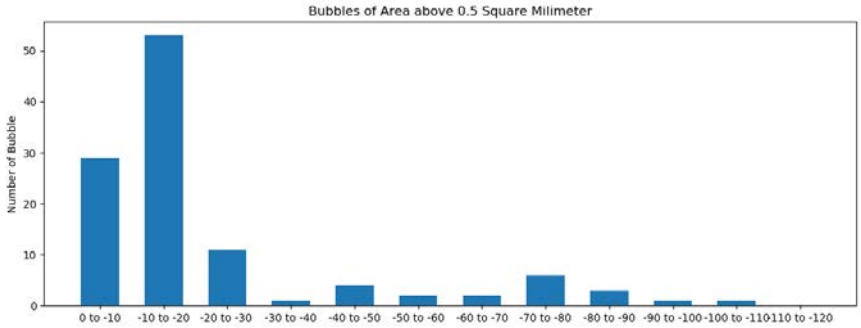

Figure 4.20.: Number of air bubbles with area above 0.5 mm^2 in Core 4 (A.4), x-axis represents the depth of the core in centimeter, y-axis represents the amount of air bubbles found in the mentioned depth of the core

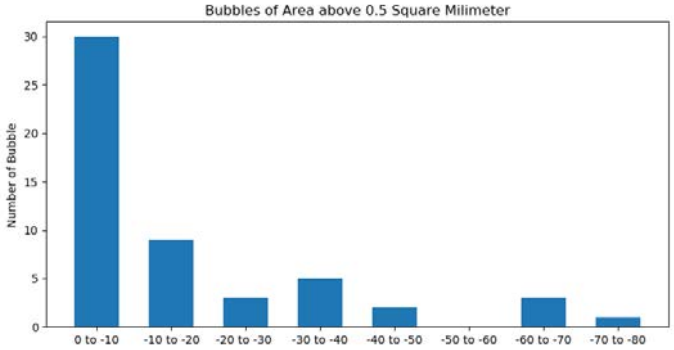

Figure 4.21.: Number of air bubbles with area above 0.5 mm^2 in Core 8 (A.4), x-axis represents the depth of the core in centimeter, y-axis represents the amount of air bubbles found in the mentioned depth of the core

5. Summary and Discussion

Satellite sensors play an important role in analyzing global sea ice properties. Since 1970, researchers are using satellite sensors in order to monitor global sea ice properties that operate on microwave frequency range (from 1 GHz to 300 GHz). Global ice properties include ice drift, ice deformation, ice growth, snow depth, ice thickness, etc which can be further used to estimate global sea ice volume, fluxes, and variabilities. During the time period between the 1980s and 1990s, Satellite SLR systems were used extensively to monitor Russian sea ice (Johannessen et al. 2000; Alexandrov et al. 2000) [26]. Sea ice extent and other ice parameters are monitored using the data obtained from polar-orbiting satellites (Johannessen et al. 1992, 1995, 1999, 2005; Jackson and Apel 2004) [25]. Based on the data received from Advanced Microwave Scanning Radiometer for EOS (AMSR-E/2) remote sensing system, snow depth is retrieved over first-year ice (FYI) and multiyear ice (MYI)(Rostosky et al.,2018)[33].

Air bubble occurrence is observed in the upper layer of Arctic sea ice. The air bubble inclusions scatter microwaves and thus changes the observed brightness temperatures from microwave radiometer and radar satellites over sea ice. This affects several sea ice related retrievals i.e. snow depth on Arctic sea ice retrieval (Rostosky et al.,2018).

In this thesis, the goal is to detect the air bubbles and brine channels by using the Convolutional Neural Network and analyze the area, circularity, and density of the bubbles inside the ice cores. The data collected in this stage is later statistically analyzed in terms of the Probability Density Function.

SSD model is trained for air bubbles and brine channel detection from Arctic sea ice core images. The Performance of the trained SSD model is measured in terms of Precision Rate. An average of 86.47% precision rate is obtained from this evaluation. The detection results show that elongated brine channels are detected partially and in some cases, the brine channels are not detected at all. As brine channel detection was not satisfactory, it is not continued for further analysis. There are some reasons behind the imprecise detection of the brine channel. Firstly, the Sea ice core images (JPG) contain noises that are some-

times elongated and look like brine channels. The model detects those noises instead of brine channels. Secondly, the number of presence of brine channels is less in the samples (200×200 pixels) used for training and testing. In results, the features of the brine channel extracted during training are insufficient for detecting brine channels independently. Finally, the partial detection of brine channels is related to the shape of the samples (200×200).

The provided ice core images are approximate of size 1800×1800 pixels. As the images with higher resolution are difficult to train, the resolution was reduced to 200×200 pixels. During the initial evaluation of the model using high-resolution images, it randomly detected big rectangles which contained a few bubbles or didn't even contain a single one. Later, with the 200×200 pixel images a significant improvement is observed as the model detected air bubble more accurately. As they are small and singular entities, this approach produced better results for air bubbles. On the other hand, brine channels are continuous, elongated entities that cover more than one 200×200 pixel area. Therefore, it resulted only in partial detection of the channel.

The convolution results on a 200×200 sample are not carrying forward to the neighboring 200×200 samples. Therefore, partial detection of brine channel doesn't get any relation with it's another part which is located on the neighboring samples. In the future, an algorithm or method can be studied and implemented which finds the relation between the neighboring samples independently regarding brine channel beside detection so that the partial detection problem can be solved.

All kinds of statistical information for air bubbles (i.e. Area, Circularity, Mean and Standard Deviation of Area and Circularity and Probability Density Function, etc.) are analyzed in the progress. Only the results obtained for Core 1 (A.4) is presented in detail in this report. Other Core results are briefly presented in tables and bar charts. From table 3.1, the average Circularity and average Standard Deviation of Circularity for air bubble measured are 0.48 and 0.18 respectively. M. E. Shokr and N. K. Sinha identified air bubbles on their study [34] in digital images of vertical thin sections from 5 hummocks and 5 melt pond multi-year ice core. On their investigation, air bubble circularity for hummock ice core and melt pond ice core calculated are 0.68 and 0.62 respectively and Standard Deviation of circularity for air bubble found for hummock multi-year ice core is 0.12 and for melt pond multi-year ice core is 0.19.

A. Appendix

A.1. Installation Procedure

The following open source software and libraries have to be installed and environment has to be set up for CNN model training. All of the installations, environment set up and training steps are accomplished in Linux operating system as recommended from the department administration but these steps can be done in Windows operating system as well.

Anaconda: Anaconda is an open source distribution of Python packages. It allows a user to create separate virtual environment for multiple project. Once the permission from the administrator is granted, Anaconda is installed in our Ubutu 16.04 LTS system and a virtual environment is created using the command line terminal by the following command.

```
conda create -n tensorflow pip python=3.5
```

By this command, a virtual environment named "tensorflow" is created inside the Anaconda for Python version 3.5.Once an environment is successfully created, the environment can be activated by the following command.

```
activate tensorflow
```

The following libraries and python packages are installed inside this virtual environment.

TensorFlow: TensorFlow is a set of open source libraries designed for Convolutional Neural Network. All of the libraries and functionalities are programmed in Python. Therefore inside the virtual environment, Tensorflow is installed.

```
pip install --upgrade tensorflow
```

In the case, Tensorflow is already installed inside the environment with lower version, to upgrade the Tensorflow version the following command has to be run.

```
pip install --ignore-installed --upgrade tensorflow
```

When Tensorflow is properly downloaded and installed, a good number of pre trained CNN model are stored in the installation folder. Amongst all of the pre trained models, SSD mobilenet is chosen to train for air bubble and brine channel detection because of its fast training speed as compared to others.

OpenCV: OpenCV is also a group of open source libraries and tools for 2D and 3D image processing. OpenCV provides tools and functionalities for feature extraction from images. To derive statistical information like area and circularity, OpenCV libraries are used. OpenCV is installed inside the Conda environment by the following command.

```
pip install opencv-python
```

MatplotLib: MatplotLib is a Python 2D plotting library used here for plotting Provability Density Function (PDF), Circularity, Area and Frequency of Bubble. MatplotLib is installed by following.

```
pip install matplotlib
```

Jupyter Notebook: Jupyter Notebook is a Python scripting platform which provides shell to run and test individual line of code, is installed by following command line.

```
pip install jupyter
```

Pandas: Pandas library is used to generate the TFRecord files during the training.

```
pip install pandas
```

Matlab Engine: Matlab has also a plenty of built-in tools for image processing, deriving information and feature extraction from image. In order to use Matlab tools or function or to run Matlab script in a Python framework, at first Matlab has to be installed on that system. The version of Matlab should be of version 2014B or above with Pyhton 2.7 or 3.4 or 3.5.

In our case, Matlab 2016A was installed in the system. A *setup.py* can be found in the root directory of Matlab. Matlab root directory is identified by running the following command in Matlab command window:

```
>> matlabroot
```

Then in the following directory, *setup.py* is located.

```
\\- matlabroot - \\ extern\engines\python
```

This *setup.py* has to be run inside the Conda virtual environment which is already created named "tensorflow". After that, *matlab.engine* has to be imported in the Python script in order to use Matlab tools.

Other Libraries: Other subsidiary libraries which are installed as well inside the environment are Pillow, lxml, cython and so on by the following commands.

```
pip install pillow
pip install lxml
pip install cython
```

A.2. Annotation Tool

The application or tool used for the annotation Air bubble and Brine Channel is called *labelImg*, is downloaded and installed from [36]. In figure A.1, the interface of *labelImg* is shown. *labelImg* provides a plenty of tools for annotating objects like drawing rectangle, labelling the objects, editing labelled objects, saving records in xml format and so on.

In figure A.2, annotated air bubbles in labelImg tool are shown. The red coloured rectangles are the annotated air bubbles, which are labelled as *Bubble*. On other images where brine channels are present, also annotated with name *Channel*.

A.3. Configuration for Training

The configuration details for training is written in this section.

```
model {
      ssd {
```

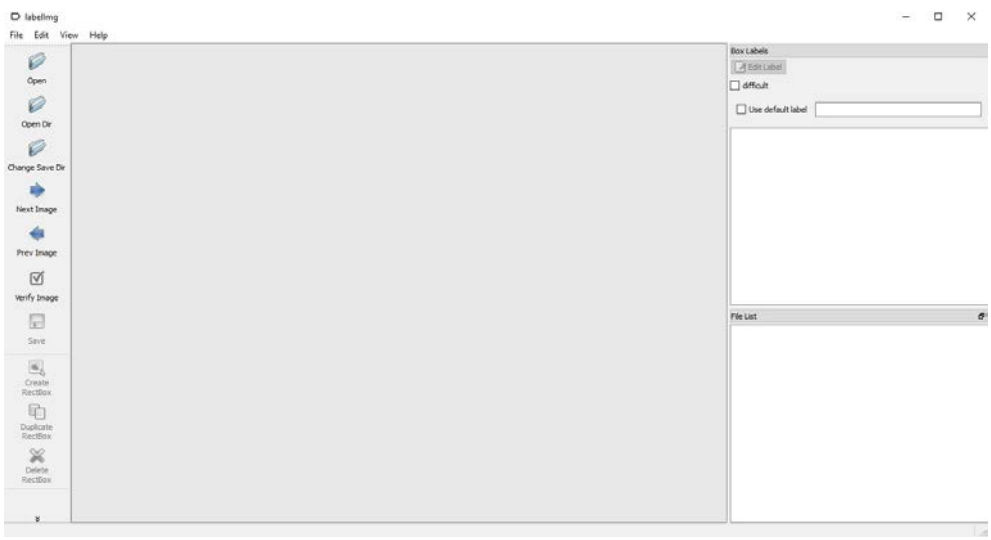

Figure A.1.: *labelImg* tool for object annotation

```
      num_classes: 2
      box_coder {
      faster_rcnn_box_coder {
              y_scale: 10.0
              x_scale: 10.0
              height_scale: 5.0
          width_scale: 5.0
      }
      }
  matcher {
    argmax_matcher {
      matched_threshold: 0.5
      unmatched_threshold: 0.5
      ignore_thresholds: false
      negatives_lower_than_unmatched: true
      force_match_for_each_row: true
    }
  }
  similarity_calculator {
    iou_similarity {
    }
```

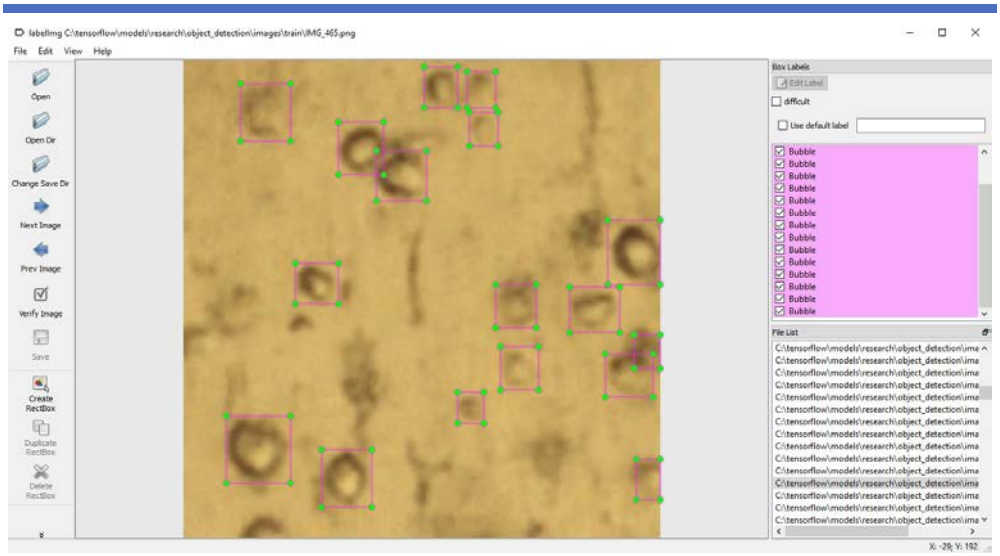

Figure A.2.: Annotated air bubbles in *labelImg* tool. The pink color rectangle (on middle) shows the manually selected rectangles as an air bubble. The right side (pink color area) shows the label of the class.

```
  }
  anchor_generator {
    ssd_anchor_generator {
      num_layers: 6
      min_scale: 0.2
      max_scale: 0.95
      aspect_ratios: 1.0
      aspect_ratios: 2.0
      aspect_ratios: 0.5
      aspect_ratios: 3.0
      aspect_ratios: 0.3333
    }
  }
  image_resizer {
    fixed_shape_resizer {
      height: 200
      width: 200
    }
  }
```

```
box_predictor {
  convolutional_box_predictor {
    min_depth: 0
    max_depth: 0
    num_layers_before_predictor: 0
    use_dropout: false
    dropout_keep_probability: 0.8
    kernel_size: 1
    box_code_size: 4
    apply_sigmoid_to_scores: false
    conv_hyperparams {
      activation: RELU_6,
      regularizer {
        l2_regularizer {
          weight: 0.00004
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.03
          mean: 0.0
        }
      }
      batch_norm {
        train: true,
        scale: true,
        center: true,
        decay: 0.9997,
        epsilon: 0.001,
      }
    }
  }
}
feature_extractor {
  type: 'ssd_mobilenet_v1'
  min_depth: 16
  depth_multiplier: 1.0
  conv_hyperparams {
    activation: RELU_6,
    regularizer {
```

```
        l2_regularizer {
          weight: 0.00004
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.03
          mean: 0.0
        }
      }
      batch_norm {
        train: true,
        scale: true,
        center: true,
        decay: 0.9997,
        epsilon: 0.001,
      }
    }
  }
  loss {
    classification_loss {
      weighted_sigmoid {
      }
    }
    localization_loss {
      weighted_smooth_l1 {
      }
    }
    hard_example_miner {
      num_hard_examples: 3000
      iou_threshold: 0.99
      loss_type: CLASSIFICATION
      max_negatives_per_positive: 3
      min_negatives_per_image: 0
    }
    classification_weight: 1.0
    localization_weight: 1.0
  }
  normalize_loss_by_num_matches: true
  post_processing {
```

```
      batch_non_max_suppression {
        score_threshold: 1e-8
        iou_threshold: 0.6
        max_detections_per_class: 300
        max_total_detections: 500
      }
      score_converter: SIGMOID
    }
  }
}

train_config: {
  batch_size: 12
  optimizer {
    rms_prop_optimizer: {
      learning_rate: {
        exponential_decay_learning_rate {
          initial_learning_rate: 0.0002
          decay_steps: 800720
          decay_factor: 0.95
        }
      }
      momentum_optimizer_value: 0.9
      decay: 0.9
      epsilon: 1.0
    }
  }
  fine_tune_checkpoint: "/home/m_alam/tensorflow1/models/research
  /object_detection/ssd_mobilenet_v1_coco_11_06_2017/model.ckpt"
  from_detection_checkpoint: true

#Note: The below line limits the training process to 300K
#steps, which we empirically found to be sufficient enough
#to train the pets dataset. This effectively bypasses the
#learning rate schedule (the learning rate will never
#decay). Remove the below line to train indefinitely.

  num_steps: 300000
  data_augmentation_options {
    random_horizontal_flip {
```

```
    }
  }
  data_augmentation_options {
    ssd_random_crop {
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "/home/m_alam/tensorflow1/models/research
/object_detection/train.record"
  }
  label_map_path: "/home/m_alam/tensorflow1/models/research
/object_detection/training/labelmap.pbtxt"
}

eval_config: {
  num_examples: 2000
  # Note: The below line limits the evaluation process to 10 evaluations.
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "/home/m_alam/tensorflow1/models/research
/object_detection/test.record"
  }
  label_map_path: "/home/m_alam/tensorflow1/models/research
/object_detection/training/labelmap.pbtxt"
  shuffle: false
  num_readers: 1
}
```

A.4. Sea Ice Core Information

Core 1 Images of Core 1 ("2015_01_26_MYI.zip", Date : 26.01.2015) are following (listed from the top of the core to bottom in sequence):

T01_IMG_0959.jpg depth from Sea Label 0 cm to −05 cm
T02_IMG_0960.jpg depth from Sea Label −05 cm to −14 cm
T03_IMG_0964.jpg depth from Sea Label −14 cm to −23 cm
T04_IMG_0969.jpg depth from Sea Label −23 cm to −34 cm
T05_IMG_0974.jpg depth from Sea Label −34 cm to −40 cm

Core 2 Images of Core 2 ("2015_01_22_FYI.zip", Date : 22.01.2015) are following (listed from the top of the core to bottom in sequence):

IMG_0587_T1.jpg depth from Sea Label 0 cm to −10 cm
IMG_0587_T1−1.jpg depth from Sea Label −10 cm to −20 cm
IMG_0591_T3−1.jpg depth from Sea Label −20 cm to −30 cm
IMG_0596_T4−1.jpg depth from Sea Label −30 cm to −40 cm
IMG_0598_T5−1.jpg depth from Sea Label −40 cm to −50 cm
IMG_0600_T6−1.jpg depth from Sea Label −50 cm to −60 cm
IMG_0605_T7−1.jpg depth from Sea Label −60 cm to −70 cm
IMG_0607_T8−1.jpg depth from Sea Label −70 cm to −80 cm
IMG_0612_T9−1.jpg depth from Sea Label −80 cm to −90 cm

Core 3 Images of Core 3 ("2015_01_26_FYI.zip", Date : 26.01.2015) are following (listed from the top of the core to bottom in sequence):

1_IMG_0615−1.jpg depth from Sea Label 0 cm to −10 cm
2_IMG_0618−1.jpg depth from Sea Label −10 cm to −20 cm
3_IMG_0621−1.jpg depth from Sea Label −20 cm to −30 cm
4_IMG_0624−1.jpg depth from Sea Label −30 cm to −40 cm
5_IMG_0626−1.jpg depth from Sea Label −40 cm to −50 cm
6_IMG_0628−1.jpg depth from Sea Label −50 cm to −60 cm
7_MG_0631−1.jpg depth from Sea Label −60 cm to −70 cm
8_IMG_0635−1.jpg depth from Sea Label −70 cm to −80 cm
9_IMG_0639−1.jpg depth from Sea Label −80 cm to −90 cm
10_IMG_0643−1.jpg depth from Sea Label −90 cm to −95 cm

Core 4 Images of Core 4 ("2015_03_05_FYI_Supersite.zip", Date : 05.03.2015) are following (listed from the top of the core to bottom in sequence):

1_IMG_0661.jpg depth from Sea Label 0 cm to −10 cm
2_IMG_0665.jpg depth from Sea Label −10 cm to −20 cm
3_IMG_0667.jpg depth from Sea Label −20 cm to −30 cm
4_IMG_0672.jpg depth from Sea Label −30 cm to −40 cm
5_IMG_0670.jpg depth from Sea Label −40 cm to −50 cm
6_IMG_0672.jpg depth from Sea Label −50 cm to −60 cm
7_IMG_0674.jpg depth from Sea Label −60 cm to −70 cm
8_IMG_0677.jpg depth from Sea Label −70 cm to −80 cm
9_IMG_0679.jpg depth from Sea Label −80 cm to −90 cm
10_IMG_0681.jpg depth from Sea Label −90 cm to −100 cm
11_IMG_0683.jpg depth from Sea Label −100 cm to −110 cm
12_IMG_0685.jpg depth from Sea Label −110 cm to −115 cm

Core 5 Images of Core 5 ("2015_03_05_Lead.zip", Date : 05.03.2015) are following (listed from the top of the core to bottom in sequence):

IMG_0687.jpg depth from Sea Label 0 cm to −10 cm
IMG_0689.jpg depth from Sea Label −10 cm to −20 cm
IMG_0693.jpg depth from Sea Label −20 cm to −30 cm
IMG_0695.jpg depth from Sea Label −30 cm to −40 cm

Core 6 Images of Core 6 ("2015_05_01_thin.zip", Date : 01.05.2015) are following (listed from the top of the core to bottom in sequence):

1_IMG_0717.jpg depth from Sea Label 0 cm to −10 cm
2_IMG_0727.jpg depth from Sea Label −10 cm to −20 cm

Core 7 Images of Core 7 ("2015_05_06_thinice.zip", Date : 06.05.2015) are following (listed from the top of the core to bottom in sequence):

1_IMG_1058.jpg depth from Sea Label 0 cm to −10 cm
2_IMG_1059.jpg depth from Sea Label −10 cm to −20 cm

Core 8 Images of Core 8 ("2015_05_10_2nd coring site.zip", Date : 10.05.2015) are following (listed from the top of the core to bottom in sequence):

1_IMG_1085.jpg depth from Sea Label 0 cm to −10 cm
2_IMG_1084.jpg depth from Sea Label −10 cm to −20 cm
3_IMG_1083.jpg depth from Sea Label −20 cm to −30 cm
4_IMG_1082.jpg depth from Sea Label −30 cm to −40 cm

```
5_IMG_1081.jpg depth from Sea Label -40 cm to -50 cm
6_IMG_1080.jpg depth from Sea Label -50 cm to -60 cm
7_IMG_1079.jpg depth from Sea Label -60 cm to -70 cm
8_IMG_1077.jpg depth from Sea Label -70 cm to -75 cm
```

A.5. Hardware Information

```
System name     : exzellnc021
Linux version   : Ubuntu 16.04.1 LTS
Memory (GB)     : 128
CPU Information : 2 x Intel Xeon E5-2670, 16*2.60GHz

Memory Information:
              total        used        free      shared  buff/cache   available
Mem:            62G        598M        351M         88M         61G         61G
Swap:           55G        216M         55G
```

B. List of Figures

C. List of Tables

D. Bibliography

[1] *Architecture of Convolutional Neural Networks (CNNs) demystified.* https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/. Accessed: 2019-03-14.

[2] *Artificial Neuron Networks(Basics)—Introduction to Neural Networks.* https://becominghuman.ai/artificial-neuron-networks-basics-introduction-to-neural-networks-3082f1dcca8c. Accessed: 2019-03-14.

[3] *CNN Architecture.* https://res.mdpi.com/entropy/entropy-19-00242/article_deploy/html/images/entropy-19-00242-g001.png. Accessed: 2019-04-22.

[4] *Contours : Getting Started.* https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html. Accessed: 2019-04-20.

[5] *Faster R-CNN Explained.* https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8. Accessed: 2019-05-18.

[6] *How to Calculate Circularity.* https://sciencing.com/calculate-circularity-5138742.html. Accessed: 2019-04-22.

[7] *How To Train an Object Detection Classifier for Multiple Objects Using TensorFlow.* https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10. Accessed: 2019-04-14.

[8] *Introduction to Computer Vision.* http://ai.stanford.edu/~syyeung/cvweb/tutorial1.html. Accessed: 2019-05-12.

[9] *Neocognitron.* https://en.wikipedia.org/wiki/Neocognitron. Accessed: 2019-03-14.

[10] *Ratio of Training data and Test Data.* https://www.researchgate.net/post/Is_there_an_ideal_ratio_between_a_training_set_and_

validation_set_Which_trade-off_would_you_suggest. Accessed: 2019-05-14.

[11] *SSD object detection.* https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06. Accessed: 2019-05-18.

[12] *Training Data and Test Data.* https://www.quora.com/What-is-the-difference-between-training-data-and-testing-data. Accessed: 2019-05-14.

[13] *Understanding Hyperparameters Optimization in Deep Learning Models: Concepts and Tools.* https://towardsdatascience.com/understanding-hyperparameters-optimization-in-deep-learning-models-concepts-and-tools-357002a3338a. Accessed: 2019-05-13.

[14] Albelwi, Saleh, Mahmood und Ausif: *A framework for designing the architectures of deep convolutional neural networks.* Entropy, 19(6):242, 2017. doi = 10.3390/e19060242.

[15] Aljburi, Dalya: *Probability Density Function (PDF)*, Juli 2016. doi = 10.13140/RG.2.1.1688.9843.

[16] Bhandare, Ashwin, Bhide, Maithili, Gokhale, Pranav, Chandavarkar und Rohan: *Applications of convolutional neural networks.* International Journal of Computer Science and Information Technologies, 7(5):2206–2215, 2016.

[17] Cao, Zheng, Principe, Jose C, Ouyang, Bing, Dalgleish, Fraser, Vuorenkoski und Anni: *Marine animal classification using combined CNN and hand-designed image features.* In: *OCEANS 2015-MTS/IEEE Washington.* IEEE, 2015. pages=1–6. doi= 10.23919/OCEANS.2015.7404375.

[18] Davis, Jesse und Mark Goadrich: *The relationship between Precision-Recall and ROC curves.* In: *Proceedings of the 23rd international conference on Machine learning.* ACM, 2006. pages=233–240. doi = 10.1145/1143844.1143874.

[19] Eicken, Hajo, Rolf Gradinger, Maya Salganek, Kunio Shirasawa, Don Perovich und Matti Leppäranta: *Field Techniques for Sea-Ice Research.* Januar 2010, ISBN 9781602230590.

[20] Fan, Q., L. Brown und J. Smith: *A closer look at Faster R-CNN for vehicle detection*. In: *2016 IEEE Intelligent Vehicles Symposium (IV)*, June 2016. pages:124-129. doi: 10.1109/IVS.2016.7535375.

[21] Fukushima und Kunihiko: *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*. Biological cybernetics, 36(4):193–202, 1980. doi=10.1109/TSMC.1983.6313076.

[22] Granskog, Mats A, Ilker Fer, Annette Rinke und Harald Steen: *Atmosphere-Ice-Ocean-Ecosystem Processes in a Thinner Arctic Sea Ice Regime: The Norwegian Young Sea ICE (N-ICE2015) Expedition*. Journal of Geophysical Research: Oceans, 123(3):1586–1594, 2018. doi=10.1002/2017JC013328.

[23] Hozo, Stela Pudar, Benjamin Djulbegovic und Iztok Hozo: *Estimating the mean and variance from the median, range, and the size of a sample*. BMC medical research methodology, 5(1):13, 2005.

[24] Hubel, David H, Wiesel und Torsten N: *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex*. The Journal of physiology, 160(1):106–154, 1962.

[25] Jackson, Christopher R. und John R. Apel: *Synthetic aperture radar: marine user's manual*. 2004.

[26] Johannessen, OM, AM Volkov, LP Bobylev, VD Grischenko, S Sandven, LH Pettersson, VV Melentyev, V Asmus, OE Milekhin, VA Krovotyntsev *et al.*: *ICEWATCH-Real-time sea ice monitoring of the Northern Sea Route using satellite radar: A Cooperative Earth Observation Project between the Russian and European Space Agencies*. Earth Observation and Remote Sensing, 16(2):257–268, 2000.

[27] Liu, Wei, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng Yang Fu und Alexander C. Berg: *SSD: Single Shot MultiBox Detector*. In: *European Conference on Computer Vision (ECCV)*, 2016. https://link.springer.com/chapter/10.1007/978-3-319-46448-0_2.

[28] Markus, Thorsten, Cavalieri und Donald J: *Snow depth distribution over sea ice in the Southern Ocean from satellite passive microwave data*. Antarctic sea ice: physical processes, interactions and variability, 74:19–39, 1998. doi= 10.1029/AR074p0019.

[29] O'Shea, Keiron, Nash und Ryan: *An Introduction to Convolutional Neural Networks*. ArXiv e-prints, November 2015. https://arxiv.org/abs/1511.08458.

[30] Qian, Rongqiang, Bailing Zhang, Yong Yue, Zhao Wang und Frans Coenen: *Robust chinese traffic sign detection and recognition with deep convolutional neural network*. In: *2015 11th International Conference on Natural Computation (ICNC)*. IEEE, August 2015. DOI: 10.1109/ICNC.2015.7378092.

[31] Ren, Shaoqing, Kaiming He, Ross Girshick und Jian Sun: *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39, Juni 2015. doi = 10.1109/TPAMI.2016.2577031.

[32] Rösel, Anja, Polona Itkin, Jennifer King, Dmitry Divine, Caixin Wang, Mats A Granskog, Thomas Krumpen und Sebastian Gerland: *Thin sea ice, thick snow, and widespread negative freeboard observed during N-ICE2015 north of Svalbard*. Journal of Geophysical Research: Oceans, 123(2):1156–1176, 2018. doi=10.1002/2017JC012865.

[33] Rostosky, Philip, Gunnar Spreen, Sinead L Farrell, Torben Frost, Georg Heygster und Christian Melsheimer: *Snow Depth Retrieval on Arctic Sea Ice From Passive Microwave Radiometers-Improvements and Extensions to Multiyear Ice Using Lower Frequencies*. Journal of Geophysical Research: Oceans, 123(10):7120–7138, 2018. doi=10.1029/2018JC014028.

[34] Shokr, Mohammed und Nirmal K Sinha: *Arctic Sea Ice Microstructure Observations Relevant to Microwave Scattering*. ARCTIC, 47:265–279, Januar 1994. doi = 10.14430/arctic1297.

[35] Specht, Donald: *Probabilistic Neural Networks. Neural Networks, 3:109-118*. Neural Networks, 3:109–118, Januar 1990. doi = 10.1016/0893-6080(90)90049-Q.

[36] Tzutalin: *LabelImg Tool*. https://github.com/tzutalin/labelImg. Accessed: 2019-04-14.