Iceberg/Ship Classification from SAR Satellite Imaging Using Convolutional Neural Networks

28 Jan 2017 Borislav Sabev s4726863

Automatic recognition of drifting icebergs from satellite imaging is important task for safe naval activities (Yu et al., 2012). Task of the current project was iceberg/ship classification for dual-polarimetric SAR imaging. This problem was tackled with an ensemble of convolutional neural networks (ConvNet). ConvNets are build using the DenseNet architecture (Huang et al., 2017) - efficient model that performs better than other vision network with far fewer parameters. The ConvNets are trained in the supervised fashion, then fine-tuned using the semi-supervised technique of pseudo-labeling(Lee, 2013). Ensemble averaged the predictions of 5 DenseNets with varying depth and width. The best performing single model's prediction have 0.29 log-loss with the true labels, which is better than the ensembles predictions with log-loss of 0.32. Pseudo-labeling worsened the result and it is not used on the models before computing the final loss.

Introduction

Drifting icebergs are a treat to naval activities in the northern part of the globe and localising them as soon as possible is vital for safe working conditions at the ocean (Yu et al., 2012). Observations from ships can provide detailed account of a drifting iceberg, but it is spatially and temporally limited (Mazur et al., 2017). Satellite imaging has proven to be useful tool in marine surveillance (Yu et al., 2012). Synthetic Aperture Radar (SAR) is a type of satellite imaging that is independent of weather conditions and lighting (Zakhvatkina et al., 2017), giving relatively consistent results over different spatial ares and in different times. SAR data can be used for automatic detection of icebergs in open ocean (Yu et al., 2012; Zakhvatkina et al., 2017). Multi-polarimetric SAR imaging includes 4 different channels HH (transmit horizontally/receive horizontally), HV (transmit horizontally/receive vertically), VH and VV (Ye et al., 2012). Different combination of channels are appropriate for different tasks, such as estimating ice thickness, segmenting ice from water, and ship detection (ibid). There exists accurate, efficient algorithms for segmenting icebergs in open ocean using dual-polarimetric SAR, consisting only of HH and HV channels, such as presented in Ye et al. (2012) and Zakhvatkina et al. (2017). However, one of the cons of SAR is low resolution which makes ship detection difficult (Heiselberg et al., 2017). The ice/water segmentation algorithms can be confused by ships, which leads to many false alarms, hence the need to better iceberg/ship discrimination form SAR data.

Mazur et al. (2017) develop a iceberg segmentation algorithm that separates everything that is not iceberg based on the HH channel of SAR data in addition to temperature and wind speed records of the imaged area. Their approach includes preprocessing the HH band with speckle noise filtering, then using OBIA based detection algorithm to segment iceberg from non-iceberg parts (ibid). Bentes et al. (2016) solve specifically the iceberg/ship discrimination problem, training a classifier to detect if an iceberg or ship is present in a satellite image. They use an X-band high resolution image of SAR data as input to a convolutional neural network (ConvNet). Their approach was significantly better than a baseline SVM, with the ConvNet achieving average F1 score of 0.97.

The satellite company C-CORE have been using computer vision algorithms from satellite data for surveillance in the past 30 years (kaggle.com). Their dataset for iceberg ship discrimination is provided as part of a Kaggle challenge to improve the current state-of-the-art in detecting drifting icebergs and differentiating them from ships. They provide dual-polarimetric SAR images from HH and HV bands. The problem is similar to the one solved by Bentes et al. (2016) the difference being that Bentes et al use images with much higher resolution and quality. ConvNets for more than 10 years now have been the state-of-the-art in computer vision (Lecun et al., 2010, Gu et al., 2017) and keep developing at fast rate. Despite different quality of the currently provided data by C-CORE and data used by Bentes et al. (2016) ConvNets are the best algorithm for object classification from image data (Gu et al., 2017) and can be used for iceberg/ship discrimination. They are excellent in learning image features on a multiple levels of hierarchy.

The research question of this project is to implement model using ConvNets that achieves state-of-the-art result in the C-CORE dataset as compared to the Kaggle community.

Method

Dataset

Samples in the C-CORE dataset (Statoil/C-CORE, kaggle.com) consist of HH and HV bands of the SAR imaging (fig.1), angle at which the image was taken and for the training set a label: iceberg (the positive class) or ship. Training data consists of 1604 data points, with 753 (~0.47 of the total) icebergs. In the training set there are 133 missing angles, all belonging to the ship class. Test set consists of 3424 data points plus 5000 machine generated data for the total of 8424 data points, with no missing values. The machine generated data is given to prevent hand labeling in the competition and it it does not influence scoring. According to Kaggle rules, test set labels are not revealed, but participants can submit prediction on their website and it returns the log-loss on the prediction with the true labels. Images in the two band are 75x75 pixels, values measured in dB. Angles are recorded as floating points numbers. The machine generated data is created, most likely by mistake from Kaggle, with angles with floating point precision of 8 digits, while the natural images record angle values up until the 4th significant digit. This allows for separation of the natural and machine generated image, which further permits the usage of semi-supervised learning. If natural and machine generated images are not separated in semi-supervised paradigm the model might learn useless features that are not relevant for the problem.

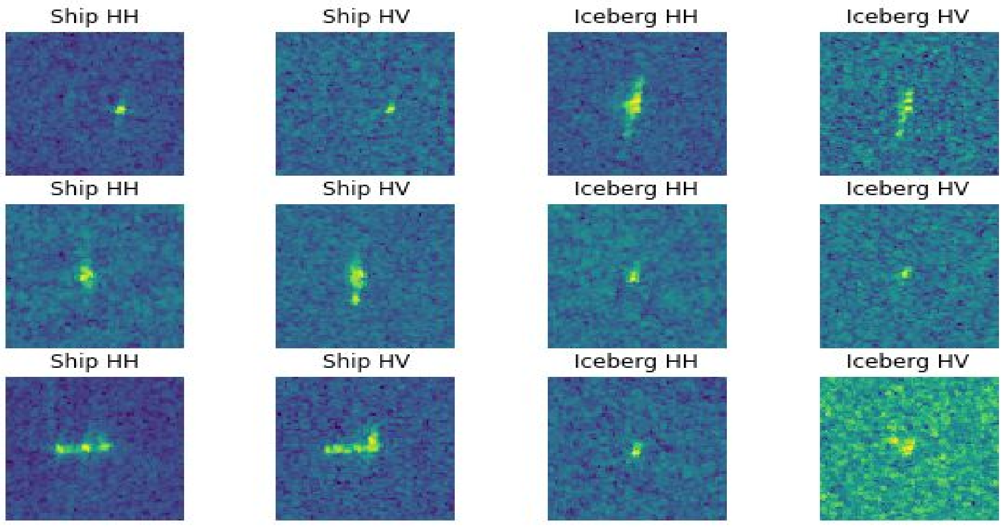

Fig. 1 Images of the 2 channels HH and HV, 3 samples from each class. The two classes generally look indistinguishable, except that iceberg is often more noisy.

Images from both bands X were separately normalized by using (X - mean(X_train)) / std(X_train), where for normalizing the validation and test data was used the mean and standard deviation of the training data to prevent leakage of information. The bands from each sample are then concatenated into a single 2 channel image, which is used as input to the model. To increase the size of the train dataset and improve generalization the training set was augmented with horizontal and vertical flips, rotation, height and width shift of 0.2 (image is shifted and 0.2 of it in height or width is replaced with noise) and shear of 0.2 (cutting 0.2 of the image and replacing with padding). This 6 options, each of which can either happen or not, theoretically produces 2^6=64 similar images from one image

Model

The current ConvNets are an implementation of the DenseNet architecture (Huang et al., 2017). DenseNet is a ConvNet achieving state-of-the-art results in 2017 on image classification with much fewer parameters than competing models (Huang et al., 2017). In the architecture of the model are introduced the dense blocks and the transition layers. Dense blocks consists of multiple layers with skip connections in between (Fig. 2b), so every layer receives and identity mapping from all other layers before it, in addition to the non-linearity from the previous layer. Defining H_l as the non-linealiarity used in layer l, H_l consist of either BatchNormalization (BN) - ReLU activation - Conv 3x3 (DenseNet) or including a bottleneck (DenseNet-B) - BN - ReLU - Conv 1x1 4k filters - BN - ReLU - Conv 3x3 k filters (Fig. 2a). The input for layer l_n is a dense block is $I_n = H(l_1, l_2, l_3, \ldots, l_{n-1})$.

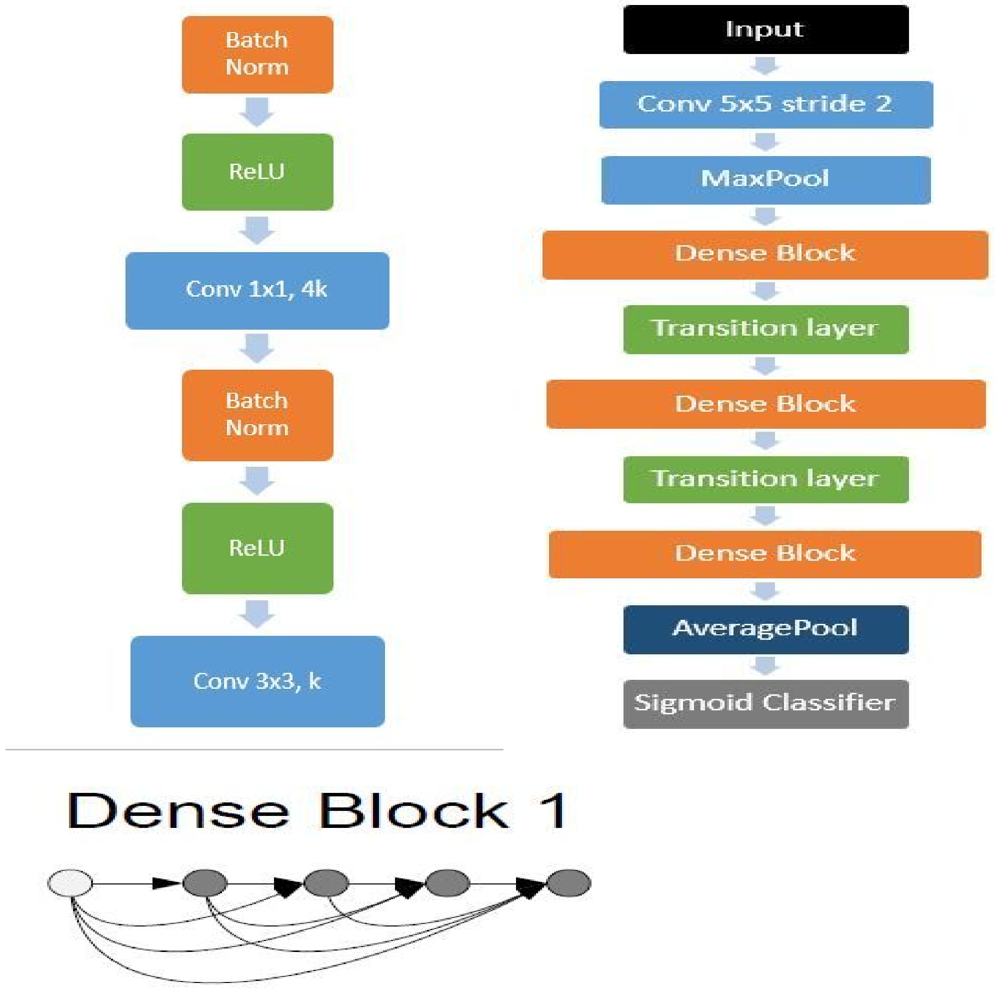

Fig 2. (a) Single bottleneck layer from dense block (left top), (b)a dense block, consisting of bottleneck layers, each giving input to all layers after it (bottom), and (c) an example architecture used in this project consisting of multiple dense blocks.

Skip connections were first implemented in ResNet (He et al., 2015), demonstrating that the gradient backpropagates more efficiently to all layers, allowing deeper networks to be trained without optimization problems. The bottleneck version of the layer uses a Conv 1x1 with 4k filters where k is the number of filters in Conv 3x3. The effectiveness of bottleneck convolutions in reducing parameters while improving performance was found from previous architectures such as Inception (Szegedy et al., 2014). Between each 2 dense blocks are placed transition layers - 1x1 convolution, ReLU activation and average pooling. 1x1 convolutions are a computationally efficient way of reducing the number of feature maps, while keeping the learned features the same. The convolution is the transition layers returns theta*k feature maps, where theta is the compression of the network. Theta = 1 means keep the same amount of feature maps while theta=0.5 implements a compression in a model referred to as DenseNet-C. The model takes as input an image and in the first 2 layers 7x7 Conv, stride 2, 2k filters, and a max pooling operation reduces the feature map size 4 times in height and width. This is followed by number of dense blocks, 4 for ImageNet and 3 for CIFAR-10, CIFAR-100, SSVH (Huang et al., 2017), with transitional layers in between. The top layer of the network includes an average pooling and a softmax prediction layer which outputs the class predictions. The k filters size is defined as the growth rate of the network and L is the number of layers in the network (Huang et al., 2017). A model DenseNet-BC (bottleneck and compression) with k=12 and L=100 with 3 dense blocks has (100-4) / (2*3) = 16 layers per dense block, total 100, minus 4 for the first, top and two transitional layers divided by the number of dense blocks and by 2 convolutions 1x1 and 3x3. The filters in the last layer in each dense block is (16-1)*k or 180.

The current model is modified version of the original DenseNet (Fig 2c). Some modifications were necessary since ImageNet dataset is different that the dataset of the current study, iceberg/ship images are much smaller 224x224 vs 75x75 and it is only binary classification as opposed to 1000 classes. Firstly the 7x7 convolution at the start of the network was replaced with 5x5 convolution, because in smaller images there will be less large global features from the start. The angle data is not used. The current DenseNet uses 3 dense blocks which are more appropriate for this input size, 4 will reduce the last feature maps for 1x1, which would no longer encode spatial information. Since the current problem is a binary classification the top of the current DenseNet is replaced with by a fully connected layer with sigmoid activation function returning the probability of the positive class. The model is implemented in Python 3 using Keras (keras.io), a library for neural network computation. The DenseNet-BCs used here are shallower than reported in the original paper (Huang et al., 2017), with L < 40, because the problem requires less features to be learned and deeper network heavily overfitted. The efficiency of DenseNet-BC allows for training multiple models.

Ensemble

Ensembling predictions from multiple classifiers can have higher accuracy than any individual classifier, provided that the classifiers' errors are independent (Tan et al., 2006). Artificial neural networks are can be assembled effectively because slight differences in architecture can lead to independence in errors (ibid). Some preliminary tests were conducted to select the five best models. DenseNets with more than 40 layers are overfitting, under 20 layers are too shallow to learn anything. The growth factor k=8 was too small, while with k=20 was training very slow and it is not more accurate than k=12. On the basic of this observations five different DenseNet-BC architectures are selected varying the number of layers L, the growth factor k and the number of dense blocks (Table 1). To further ensure the independence of predictions, the each of the networks can be trained on different parts of the training data data. Training data is split into 5 equal parts with a stratified Kfold split, keeping original fraction of positive class from the training set in all splits. Every model was trained on a different set of 4 folds, keeping the last of the folds for validation. This way all models can be validated, but at test time the collective ensemble has seen all labeled data. The ensemble predictions are a average of the predictions of all model.

Training

Each epoch consists of going over all training examples with 64 samples per batch, validating and recording the current train and validation accuracy. Models are trained with Adam optimizer with a starting learning rate of 10^{-3}, where learning rate is reduced 10 times for every 5 epochs the learning rate is not improved. The starting learning rate is chosen to be 10 times more time than the recommended for the optimizer because using Batch Normalization layers allows for a higher learning rate (Ioffe et al., 2015) thus faster training.

The test dataset, counting only natural images, is about twice as large as the training data (3424 test examples, 1604 train). This large amount of test data can be used to further increase the performance of the supervised DenseNet by fine-tuning in a semi-supervised fashion. To make use of the large amount of test data a semi-supervised technique known as pseudo-labeling (Lee, 2013) is employed here. In pseudo-labeling there is a classifier trained supervised on the labeled data, then used to predict labels of unlabeled data (Lee, 2013). The unlabeled data then is used to train the classifier, while the predicted labels act as real labels. This technique can potentially increase test score by enabling the classifier to learn from more data (Lee, 2013). The danger is that with wrong labels the test score could decrease. In the current project pseudo-labeling was implemented by predicting all samples in the test set, then selecting for training only the samples that are predicted with at least 95% confidence, <0.05 for the negative class and >0.95 for positive class. Test data with pseudo labels were then merged with the corresponding training folds for each model and shuffled. If the pseudo-labeling stage didn't include the training data, test that could move away the gradient from a place of better local optimum. The model then is trained for 1 epoch, and on the next epoch test samples are predicted again, when new pseudo labels, repeating the procedure for 5 times in total. Results with and without pseudo-labeling are compared to investigate how beneficial is the approach to this problem. Final evaluation of the models and the ensemble is the log-loss on the test data.

Results

All models achieved a plato in validation accuracy in about 60 epochs (Fig.3). Four of the models reached ~85% validation accuracy, with model L=32,k=16 the only one that stayed at ~80%. The falls in validation accuracy are signs of overfitting, which was resolved by scheduling reduction of the learning rate.

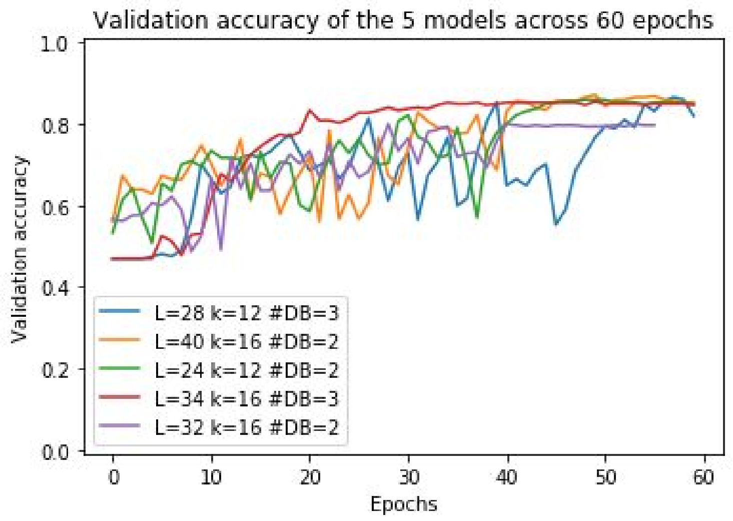

Fig. 3 Validation accuracy for all models during training. L is number of layers, k is growth factor and #DB is the number of dense blocks.

After training was completed all models were used to predict the test data and record a submission to Kaggle. The test log-loss is in Table 1. The best performing single model was DenseNet-BC{k=16, L=40}with a log loss of 0.2858. The ensemble prediction was computed by averaging all predictions. The ensemble of all models had a log-loss of 0.32, larger than the best single model. Even ensembling the top 3 models is still worse than the DenseNet-BC{k=16, L=40}.

Pseudo-labeling worsened the validation accuracy of all models, bringing it down to less than 65%. For that reason, the models using pseudo-labeling were not tested on the test set. After the pseudo-labeling all models except {L=28, k=12} were predicting only the negative class.

Models	Depth	# Dense Blocks	Number of parameters	Test loss
DenseNet-BC (k=12)	24	2	80k	0.3615
DenseNet-BC (k=12)	28	3	70k	0.4105
DenseNet-BC (k=16)	32	2	185k	0.4529
DenseNet-BC (k=16)	34	3	181k	0.3145
DenseNet-BC (k=16)	40	2	256k	0.2858
Ensemble (all models)				0.3210
Ensemble (top 3 models)				0.2918

Table 1. All tested models with their depth and number of parameters and the corresponding test loss. Best model in blue.

Discussion

Five DenseNet models were compared for log loss on the test data. Surprisingly, models with very similar validation accuracy (1st and 5th model from Table. 1) 85% and 85.4% have a significant difference in test log-loss 0.36 vs 0.29. The best performing model is the deepest one with the most parameters, which makes sense, since more parameters means fitting more complex features. However relationship between depth, parameters and performance is not that simple - the worst model {L=32,k=16} is deeper and has twice as many parameters and the third scoring model {L=24,k=12}. Test results after the pseudo-labeling were not reported, because the validation accuracy was drastically decreased for all models. Even with using only examples that the model is confident about, most pseudo-labels must have been assigned wrong for the training not to work. It is possible that it is a case of the model having high confidence about the wrong thing as reported in Nguyen et al. (2015). However, having that much model worsening by wrong labeling is suspicious, because the best performing model {L=40,k=16} has a relatively low log-loss, which means that is shouldn't have that many wrong high confidence classifications. In Lee (2013) a ConvNet with a 23% error rate benefits from pseudo-labeling, which reduces the error rate to to 16%. Given this result I find it unexplainable why the models were performing so bad after using pseudo-labeling. The ensemble of the five models is not satisfactory, because the two low-performing models are driving down the result. The ensemble with the top 3 models is performing better than the ensemble from all models, but still less than the best single model. The less modes are used, the less there is benefit from ensembling.

Conclusion

In conclusion the problem of iceberg/ship classification was tacked as any object classification from an image problem, with convolutional neural networks. The outcome of the project was quite poorer than expected. The latest results were submitted after the end of the competition, but if the competition was still on, my best result would bring me to 2498 place out of 3343 teams. In hindsight, it was probably a better idea to train deeper models even though they heavily overfit as some weight regularization and dropout could have helped. The shallower models that were used just did not learn enough features to score better. Ensembling the results did not work because of poor and too few models. The pseudo-labeling gave terrible, unexplainable results, most likely because the models were not accurate enough to benefit from from assigning their own labels to the data. Another point worth mentioning is that even with the most efficient ConvNet architecture, it is still not reasonable to train on a CPU. Lack of GPU access was one of the reasons to use shallower models, which at the end did not perform satisfactory.

For further research it could have been useful to the model to use the incidence angle in preprocessing the HH channel as in Zakhvatkina et al. (2017). Another possible use of the incidence angle is to be added to the fully-connected layer, but then the training examples with missing angle values need to be excluded. Instead of validating model on a single split, all models could be cross-validated on every split and at prediction time to pick the best model of each split. The reason it is not used here is that it will take 5 times more time to train and the current training takes on a single CPU (Intel i3, 2GHz) over 24 hours. Another possibility to improve the score it to train more models for the ensemble, which was again not used because of the expensive computations involving training neural networks. The current ensemble used a simple averaging of the results. This could be improved upon by concatenating the output of all networks, putting fully-connected (FC) layer for classification, freezing the trained networks and training only the FC layer between the networks outputs and the final output. This way the FC layer can learn how important is the vote of each network. Adding two FC layers instead of one can learn more complicated relationships between the votes, but also can overfit. With better performing models, pseudo-learning might be significantly more useful that it was in the current project.

Reference

He, K., Zhang, X., Ren, S., Sun, J. (2015) Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.03385

Lee, D. (2013). Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks. https://www.researchgate.net/publication/280581078_Pseudo-Label_The_Simple_and_Efficient_Semi-Supervised_Learning_Method_for_Deep_Neural_Networks

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, Vanhoucke, V., Rabinovich, A. (2014). Going Deeper with Convolutions. https://arxiv.org/abs/1409.4842

Huang, G., Lui, Z., van der Maaten, L. (2017). Densely Connected Convolutional Networks. https://arxiv.org/abs/1608.06993

Tan, P.-N., Steinbach, M., & Kumar, V. (2006). *Introduction to data mining*. Boston: Pearson Education, Inc.

Ioffe, S., & Szegedy, C. (2015). Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://arxiv.org/pdf/1502.03167.pdf

Zakhvatkina, N., Korosov, A., Muckenhuber, S., Sandven, S., & Babiker, M. (2017). Operational algorithm for ice – water classification on dual-polarized RADARSAT-2 images, 33–46. https://doi.org/10.5194/tc-11-33-2017

Yu, P., Qin, A. K., & Clausi, D. A. (2012). Feature extraction of dual-pol SAR imagery for sea ice image segmentation. https://doi.org/10.5589/m12-028.

Heiselberg, P., & Heiselberg, H. (2017). Ship-Iceberg Discrimination in Sentinel-2 Multispectral Imagery by Supervised Classification https://doi.org/10.3390/rs9111156

Bentes, C., Frost, A., Velotto, D., Tings, B. (2016). Ship-Iceberg Discrimination with Convolutional Neural Networks in High Resolution SAR Images.

Mazur, A. K., Wåhlin, A. K., & Kr, A. (2017). An object-based SAR image iceberg detection algorithm applied to the Amundsen Sea, *Remote Sensing of Environment 189*, 67–83.https://doi.org/10.1016/j.rse.2016.11.013

Lecun, Y., & Kavukcuoglu, K. (2010). Convolutional Networks and Applications in Vision. http://yann.lecun.com/exdb/publis/pdf/lecun-iscas-10.pdf

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., & Shuai, B. (2017). Recent Advances in Convolutional Neural Networks, 1–38.
https://arxiv.org/pdf/1512.07108.pdf

Nguyen A, Yosinski J, Clune J. (2015) Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images. In Computer Vision and Pattern Recognition (CVPR '15), IEEE, 2015.

Statoil/C-CORE Iceberg Classifier Challenge: Ship or iceberg, can you decide from space? https://www.kaggle.com/c/statoil-iceberg-classifier-challenge

Keras: high level neural networks API https://keras.io/

Appendix

All code used in the project can be found on
https://github.com/BobiSabev/IcebergShip

Code:

`preprocess_data.py` - loads the original json files, fills missing values, fixes data types, extracts the is_natural feature, saves everything into bcolz for fast loading

`visualization.py` - for the EDA

`densenet.py` - defines a class for DenseNet as described in the original paper

`train_ensemble.py` - contains all the code for loading data, normalization, creating models, training, pseudo-learning, and writing submission for Kaggle

Other:

In the folder submission are all the submission posted on Kaggle to get the final test loss. The folder model contains weights of trained models.