Object Segmentation in Overhead Imagery using Capsule Networks

Ruth Wijma

Supervisor: Dr. Mark Hoogendoorn

Vrije Universiteit Amsterdam

Abstract. Convolutional Neural Networks (CNN) are currently the state of the art in both image classification and recognition. This does not mean that they don't suffer from drawbacks, like rotational and translational invariance. A new approach has been introduced to counter some of these, resulting in more predictive power while still using the same amount of data. This approach is called a Capsule Network [1]. This paper shows the result of applying this model to satellite imagery, and compares the results to a standard CNN.

Keywords: Capsule Network· Convolutional Neural Network · Object Recognition · Overhead Imagery

Table of Contents

1 Introduction

There have been lots of examples of utilizing the incredible power of modern processors to the field of computer vision. The start of this branch of science started with classifying pictures into one of the possible classes. There are the fields of object detection and recognition, which continued on this initial problem, where the goal is to see if there is an object in the image, and to also determine its class respectively.

Research in the last 20 to 30 years has shown that this problem can be solved, given enough data, by using some form of a Convolutional Neural Network (CNN). This model has been applied to binary classification problems, such as classifying an image as being either a cat or a dog [2], or multiclass classification: classifying handwritten digits in the range of 0-9 [3], identifying objects in a live feed [4], all with reasonable accuracy.
So, what exactly can be achieved by furthering in this field of science? There is first of all the problem of having large data sets. Large data sets itself can be obtained with a relative ease, labeling each instance however can take a lot of time, which is costly for both researchers and businesses using this technology. It is therefore wanted to decrease the number of instances needed for a predictive model.

The problem stated above can be solved by creating translated copies of the original data, such as a slightly rotated or resized image. It is of course necessary to first verify if the translated copy is indeed still a part of its original class. E.g. a '+' rotated 90 deg should not be classified as a '+' but as a '×'. But what if the set of possible classes is large, and just a subset of these object classes are invariant under translation? What if parts of these objects still remain within their respective class after translation, but other parts do need to stay fixed? These questions can also be asked about the set of possible translations, such as shifting, rotating, resizing or cropping. This means that using this approach either results in too few representations of the objects, resulting possibly in false negatives, or too many, possible leading to too many representations, and thus to the model output containing false positives. Besides this, creating all of these extra instances is an extra step in setting up a model and creates extra computational overhead, because of the extra time needed during training.

This problem inherent of CNNs is called *translational invariance* [5]. Hinten et al. proposes a method for solving this problem, by using a novel neural network architecture, called a Capsule Network [6]. This network is focused on learning possible representations of objects, by inferring a model of them.

2 Literature

Lots of methods have been proposed to solve the high-dimensional problem of object detection and segmentation. Before the era of deep learning took over, features were (semi-)handcrafted, mostly to fit a specific problem [7]. The well known Viola-Jones algorithm [8] was developed in 2001 and is still used by modern cameras to detect faces in image feeds because of its speed. Features are

extracted by first creating an integral image, which is the a cumulative sum across both axis of the image, from the top-left corner to the bottom-right. This integral image is then used to quickly calculate differences between certain regions in the image. They soon found out what were good features to use in the algorithm in terms of predictability, and created a decision process to determine if a certain region of the image was likely to be a face.

In 1998, LeCun et al. were one of the first to use convolutions as a general purpose feature detector in neural networks [9]. They compared several methods for object recognition, and found out that using these layers proved to be useful in outperforming other methods. They applied this method to 32×32 images, but found out that advances in computational theory were needed to apply this method to larger resolution images, since computational overhead was still a problem at this time.

Some 14 years later, In 2012, Krizhevsky et al. [10] proposed a novel method which used deep learning to compete in the ImageNet ILSVRC (ImageNet Large Scale Visual Recognition Competition) challenge, which outperformed all other methods. It is in essence the same model as LeCun proposed in 1998, but this time applying a model of this nature was feasible because of the use of graphical processing units. This is the first time deep learning was successfully applied to a problem of this nature and scale, outperforming all classical computer vision models using custom-made features in this competition.

Since then, lots of variations have been made to the standard CNN. GoogLeNet [11], otherwise known as InceptionV1, is one of these models, which uses Inception modules. Each module uses different sizes of convolutional filters. It also differed in other minor ways, such as the use of a different training algorithm and the use of batch normalization.

The Residual Neural Network (ResNet) [12] was the winner of the 2015 ILSVRC challenge. It is the first network of its kind using residual or *skip connections* in its architecture. These connections allow the network to preserve some information while it down samples the input. It is useful in creating a very deep structure, while keeping the complexity relatively low.

All of these methods continue to improve predictable performance as a general purpose image recognition or detection task. However, the generalizability of CNNs results in drawbacks. Most architectures generalize so well, that the resulting architecture is invariant to a lot of changes in the input, such as translational or rotational invariance [5]. This means that shifting or rotating an object in an image does not change the model output. This can be a good thing, but it can also result in false positives if the model is too general. Figure 1 shows such a false positive, and shows why it is necessary to model which invariances are possible for each object. Hinton et al. [6] proposed a network able to model objects in such a way, called a Capsule Network (CapsNet). This model is able to learn a model of an object, by learning transformation matrices which learn possible relationships between the image (the viewer) and the entity itself. This results in a model which exhibits realistic detection.

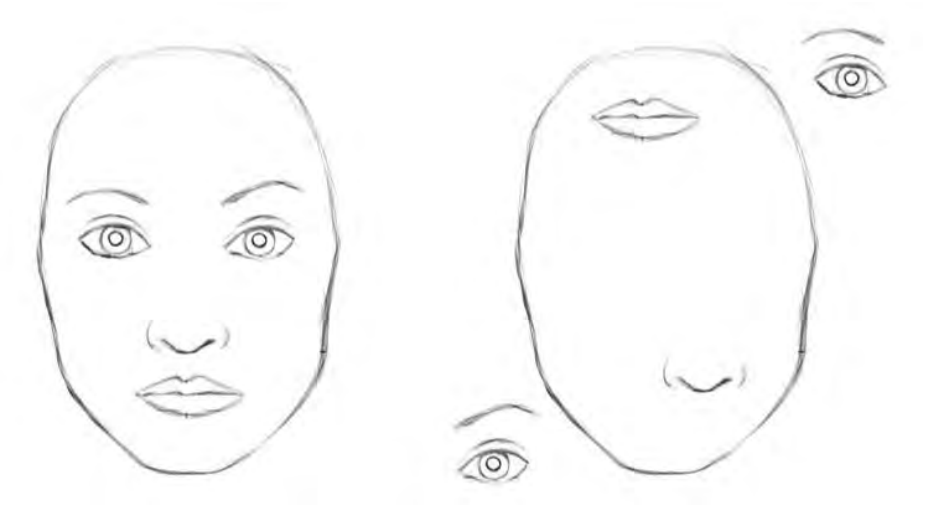

Fig. 1. Two images, both predicted to be a face, showing the problem of translational invariance. Taken from an article about the intuition behind CapsNet [13].

3 Data Description & Analysis

The data set and description are supplied by Airbus for a competition hosted on Kaggle [14]. The data itself consists of a large set of satellite images, each of them with a resolution of 768×768. Each of the pixels has three color channels, making each image effectively a $768 \times 768 \times 3$ tensor. Each entry is an integer value in the range $[0, 255]$. The data set contains a total of 192.556 images and amounts to 27.1 GB of uncompressed image data. The total number of boats present in the images is 81.723.

The label of each image is called a "mask", and is represented by a 768×768 matrix containing only zero and one entries, where all ones indicate presence of a boat. Some example images and their respective masks are given in figure 2.

The data is very unbalanced when looking at the number of boats per image, and the size of the boats itself as can be seen in figure 3. This means that conventional metrics, such as accuracy or binary cross entropy, won't work, as using them would just result in a model which predicts empty masks for each image. It is therefore useful to first down sample the images containing no boats by a high factor, such as 0.9. Since this does not prevent that images containing one or more boats will still be mostly empty, a different loss metric has to be used. The accuracy score for predicting an empty mask for each image still is around 99.6%, which shows the need for a metric that punishes false negatives more heavily.

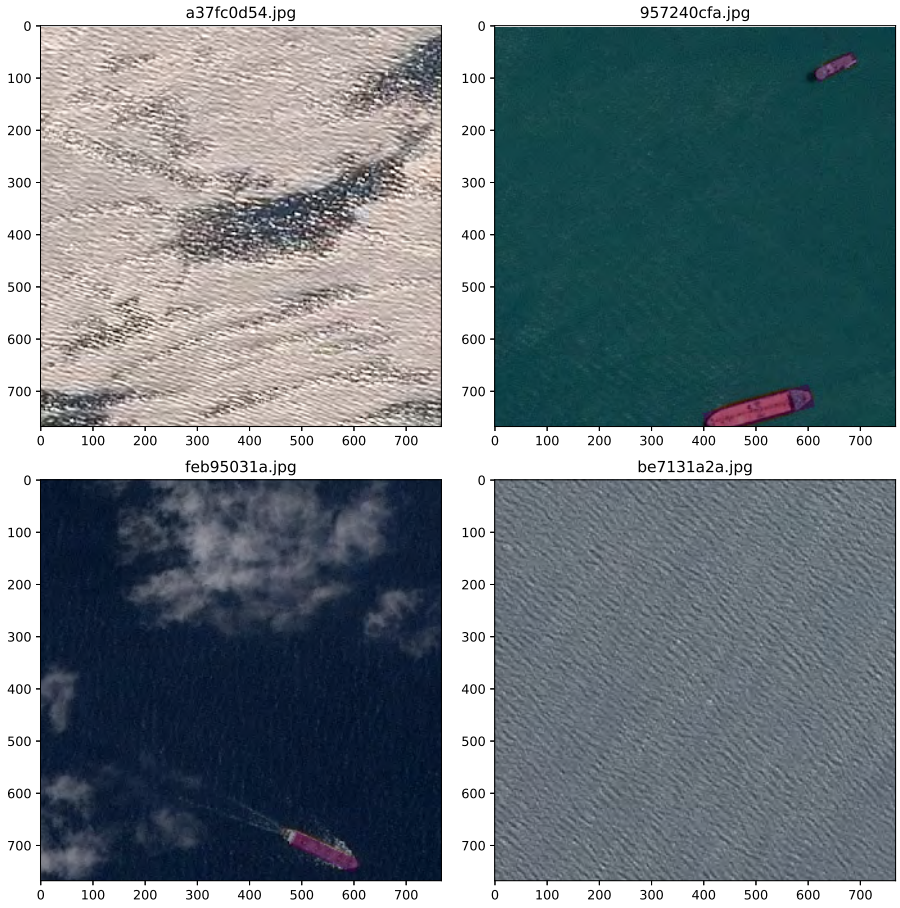

Fig. 2. Example images from the data set and their masks, shown in purple.

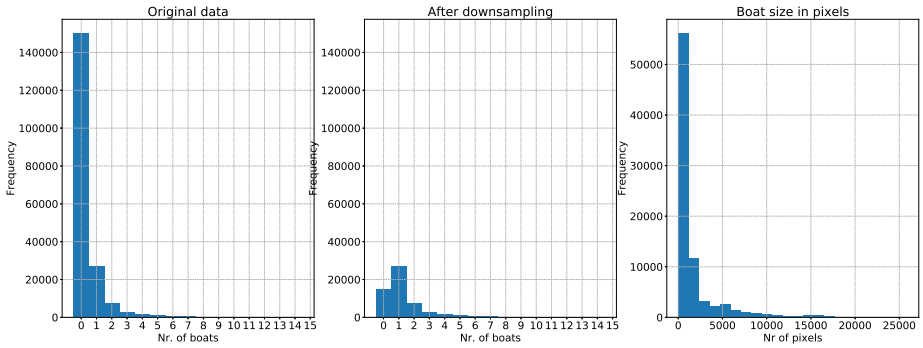

Fig. 3. Histrogram showing the distribution of the number of boats per image (left and middle figure), histrogram showing the boat size (right).

4 Methodology

4.1 Focal Loss

Focal loss [15] is an error metric used to evaluate models in which the original data has a very skewed distribution. This loss function, given in equation 1, is in essence a re-weighted cross entropy error function, which is obtained by setting the hyper-parameter $\gamma = 0$. Setting $\gamma > 0$ results in more punishment for false positives, which aids us in fitting a model to a skewed dataset.

$$FL(p_t) = -(1 - p_t)^{\gamma} \log(p_t) \tag{1}$$

4.2 Convolutional Layer

A convolutional layer is a means of learning features by creating filters. Filters slide (or convolute, hence its name) over the input. Each of the filters in the used CNN used a window size of 3×3. The filters are initialized randomly. Stride was set to one, implying that a stepsize of one was used for each filter. Zero-padding was used to make sure the dimensions of the output did not differ from the input, by adding zeros at the margins of the image. Since each filter creates a feature-map, it scales the output dimension by the number of filters. If the input is $H \times W \times D$, then the output has dimensionality $H \times W \times (D \cdot F)$.

Filters are represented by matrices, where the values can be interpreted as weights. These weights are initialized by sampling from the Uniform distribution between $-a$ and a, where

$$a = \sqrt{\frac{6}{I + O}}$$

with I the number of input units and O the number output units. This is called the Glorot uniform initializer [16].

4.3 Max-Pooling Layer

A max-pooling layer is a means of reducing the input to a lower dimension, resulting in a more robust model. Applying this layer to an input tensor results in using a convolution matrix, but this time on non-overlapping elements. A 2×2-pooling window was used. This means that each 2×2 entry of the original matrix is mapped to a single value by taking the maximum, see equation 2. Applying this layer to an input tensor X reduces the input in both dimensions by half.

$$\text{MaxPool}(X)_{i,j} = \max(X_{2i,2j}, X_{2i,2j+1}, X_{2i+1,2j}, X_{2i+1,2j+1}) \tag{2}$$

4.4 Upsampling Layer

This layer is used to increase the input dimension to a higher dimension. This is done by simply copying neighboring elements.

4.5 Dropout Layer

One of the ways in which it is possible to prevent over fitting to the data, is by using a dropout layer [17]. This layer is, just like a dense layer, fully connected, but differs in the fact that with some probability p drops out, hence its name, of the network structure, and thus disappears as an input to the next layer. This parameter p is optimized during training.

Since consistent output is preferred when the model is optimized, we want to remove any stochastic influences on the model output during testing. Therefore, the dropout layer becomes a regular dense layer, and its output weights are multiplied by p.

4.6 Capsule Layer

The capsule layer is the defining layer in the capsule network. It was introduced in the paper by Hinton et al [18]. It tries to learn a representation of an object, during training time of the network. This process is called inverse graphics, and aims at trying to understand the scene by looking at the data. This information is stored in the capsule and has, depending on the architecture, a learned value for position, rotation, scale, albedo, and other interesting aspects of the instantiating of the object.

4.7 Dynamic Routing

Whereas lower level capsules learn smaller features, such as lines, edges or blobs, higher level capsules make use of these lower level features and combine them into objects. The crucial difference between a CNN and a CapsNet lies in the fact that an alternative to Max Pooling is used to learn this connection. CapsNet makes use of the process called Dynamic Routing by Agreement, which learns

when capsules between layers agree on the input. This process forces the network to learn spatial relationships between features, so that the face in figure 1 (left) will be recognized as a face, whereas the scrambled face in figure 1 (right) will not, because the capsules in different layers do not agree on the relative positions of the elements of the face.

4.8 Convolutional Neural Network

The CNN achitecture used in this paper is a U-net [19], using several down-sampling layers, followed by multiple upsampling layers. All of these layers start with a convolutional layer with a certain number of filters, all set to a certain size. These convolutional layers are followed by a dropout layer. Layers at the same 'depth' share a skip connection. This makes it possible for the network to use detailed information, as well as the feature maps created along the way.

4.9 Capsule Network

The Capsule network architecture used in this paper [20] is created with the idea of segmentation in mind and thus only replaces the fully connected layer of the second half of the network by a upsampling capsule layers.

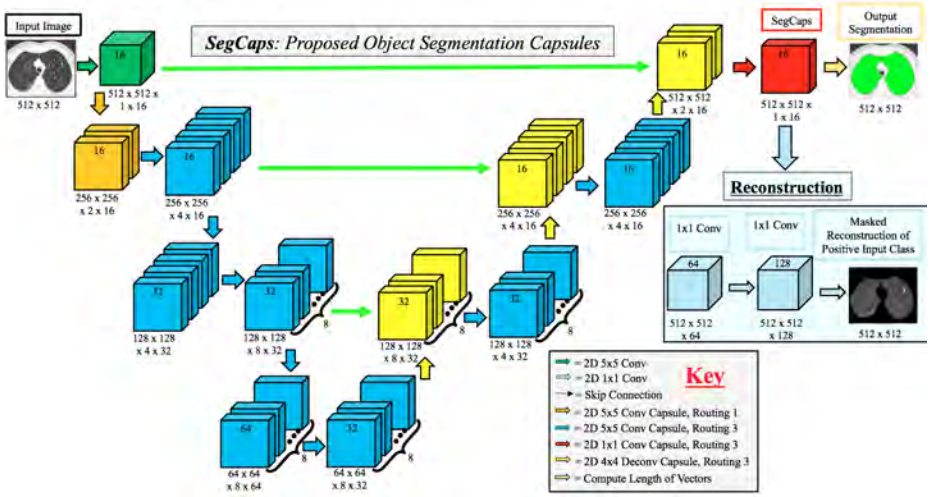

Fig. 4. SegCaps network [20]

5 Experimental Setup

Standard methods are used to prepare the data as input for the model. The subset of the data of images containing no boats is down-sampled by a factor of 90%. Each image is normalized to the interval [0,1]. The resulting data set is then split up in fixed chunks, meaning that they are static over multiple runs, ensuring no data leakage in the independent test set.

After pre-processing, the data set is split up in a train set containing 60% of the data, and a validation and test set, each taking 20% of the data. The validation set is used to tune the hyper-parameters, and the test set is used to independently measure the final model performance.

Multiple hyper-parameters were varied to improve model performance. A Bayesian framework was used to do this efficiently by imposing priors on the parameters. Sufficient subsets of the train and validation data set were used for this procedure. For the CNN, the number of layers was not contained in this set of hyper-parameters. Instead, for each setting, called an architecture, the hyper-parameter optimization was ran. This allows us to inspect the best model for each architecture, which makes it possible to inspect the impact of the complexity of the model on its performance. Similarly, hyper-parameter optimization for the Capsule Network was ran for two different architectures, namely a small variant, called Basic, and a larger one, called R3 [20].

After the set of hyper-parameters is found for each CNN architecture, the model is fitted 5 times on the full training set, and the performance of each run is measured on the test set. Statistics on each architecture are reported. A similar approach was used for the Capsule Network, this model has two architectures, and is therefore only reported twice.

The dropout parameter, D, is often used in CNNs and is mostly set to 0.5 as a starting value [17]. Since the dropout parameter is a probability, its value needs to be in [0,1]. A standard uniform distribution was used for its prior.

Another hyper-parameter of the CNN is the number of filters F. The number of filters layer grows exponentially in the number of layers for this specific architecture. For layer i, the number of nodes is $B \times 2^{i-1}$. The number of layers is defined as the number of layers of the down-sampling part of the network, since the network structure is symmetric. A similar approach was used to set the number of filters in the Capsule Network. This base size is in the range [4,8,16,32]. This base size is referred to as B.

Lastly, the learning rate parameters, λ, was varied. An loguniform prior was assumed over this parameter with parameters $\mu = \log 0.001$ and $\sigma^2 = \log 0.02$.

5.1 Evaluation

Multiple evaluation methods have been proposed to assess the quality of a model on a certain dataset regarding object recognition. The choice depends on the problem at hand, the final goal of using the model, available computational power, conveniences regarding the chosen model, and multiple others.

F_β-Score The F_β-score, as depicted in equation 3, is used to evaluate segmentation predictions as a function of its calculated number of true positives, false positives and false negatives. This metric is used often for $\beta = 2$. Setting $\beta = 1$ results in the F_1-score, otherwise known as the Dice coefficient.

Whether or not a prediction is a true positive (TP), false positive (FP) or false negative (FN) is determined by the Intersection over Union score (IoU) being larger or smaller than a predefined parameter $\alpha = 0.5$. Predictions having a ground truth match with IoU $> \alpha$ will be counted as true positives, whereas predictions without a ground truth match, i.e. IoU $< \alpha$, will be counted as false positives. False negatives are the ground truth for which IoU $< \alpha$.

$$F_\beta^\alpha = \frac{(1+\beta^2)\mathrm{TP}_\alpha}{(1+\beta^2)\mathrm{TP}_\alpha + \beta^2\mathrm{FP}_\alpha + \mathrm{FN}_\alpha} \tag{3}$$

$$F_1^\alpha = \mathrm{Dice} = \frac{2\ \mathrm{TP}_\alpha}{2\ \mathrm{TP}_\alpha + \mathrm{TP}_\alpha + \mathrm{FN}_\alpha}, \qquad F_2^\alpha = \frac{5\ \mathrm{TP}_\alpha}{5\ \mathrm{TP}_\alpha + 4\ \mathrm{FP}_\alpha + \mathrm{FN}_\alpha} \tag{4}$$

6 Results

6.1 Hyper-parameters

The resulting hyper-parameters can be found in table 1. Hyper-parameter optimization was not performed for the R3 network, since the computational complexity is still too high to perform long runs of optimization. Instead, the parameters of the Basic architecture were used again, assuming that the network works well on the same parameter settings.

We can see that the optimal learning rate λ based on these runs is low (lower than 5×10^{-3}) each time, except for the ConvNet with 3 layers. The base number of layers varies between 1 and 3. These results show that reasonable parameters were found, but that no particular pattern can be retrieved. This might mean that the hyper-parameter optimization did not converge, and that more could have been done to improve. This was however not feasible due to time constraints.

6.2 Metrics

As we can see from table 2, the CapsNet Basic architecture just slightly outperforms the ConvNet having five layers. This result should also take into account the number of parameters used in the model. The ConvNet with 4 layers uses 26k weights, whereas the 5 layer ConvNet has 46k weights. The basic variant of CapsNet uses 44k weights. This run does outperform the results of the ConvNet while also being more efficient. It should however be noted that just one run was performed for both CapsNet variants, and thus nothing can be said about its significance.

Model		λ	D	B
ConvNet	1 layer	0.0047	0.3346	3
ConvNet	2 layers	0.0083	0.0534	2
ConvNet	3 layers	0.3521	0.3521	1
ConvNet	4 layers	0.0017	0.3028	3
ConvNet	5 layers	0.0012	0.1604	2
CapsNet	Basic	0.0016	-	16
CapsNet	R3	-	-	-

Table 1. Hyper-parameters used to train the final model on the given architecture. The hyper-parameters found for the CapsNet Basic architecture were also used for the R3 architecture.

The CapsNet R3 does a lot worse than its smaller variant, but this might be a result of the huge number of parameters needed to fit the model, and that simply more time was needed for the training to converge.

It is also clear that the accuracy is not a useful metric to assess the model quality, since it is nearly uniform over all 7 model types, whereas the DICE score is significantly different.

The CapsNet model that got the best results was used to predict the object masks for 25 randomly sampled images from the test set. These results can be seen in figure 5. It can be seen that the model has difficulty predicting the bounding box correctly for large boats (top left), which might be caused by the fact that there are mostly small boats in the data set (Figure 3, right).

Model	Accuracy (%)	DICE	F_2	TP	FP	FN
ConvNet 1 layer	99.374	0.1895	0.215	2156	1241	17199
ConvNet 2 layers	99.607	0.178	0.1749	2524	6473	16 831
ConvNet 3 layers	99.634	0.1624	0.1699	2056	3910	17299
ConvNet 4 layers	99.493	0.5921	0.5787	9997	4417	9358
ConvNet 5 layers	99.704	0.7454	0.7611	12237	1241	7118
CapsNet Basic	99.8376	0.7519	0.7669	12406	1238	6949
CapsNet R3	99.598	0.6663	0.693	10209	1079	9146

Table 2. Mean results on test set for all model types.

7 Conclusion

Because of computational complexity it was not possible to fully investigate the possibilities of the Capsule Network. It does however show that promising results are possible, but more computational power is needed, or smart optimizations have to be made to the training algorithm for this particular network. The

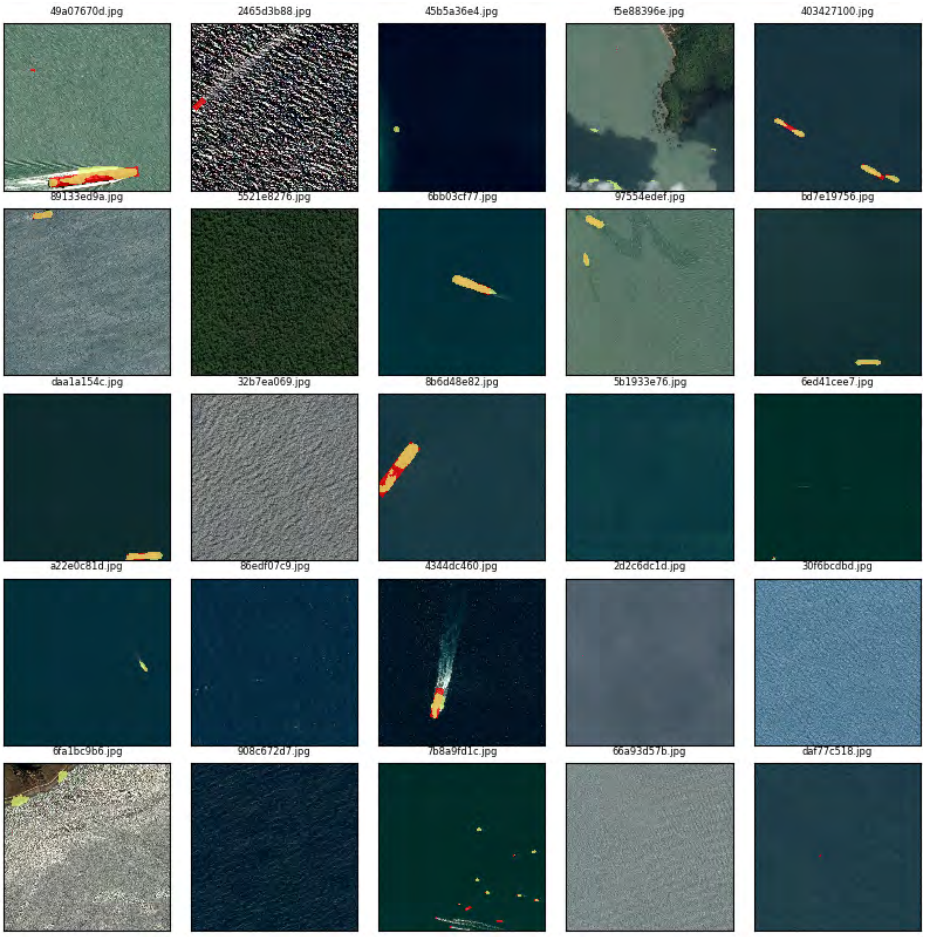

Fig. 5. Predicted object masks (yellow) and ground truth (red) for 25 randomly sampled images from the test set.

resulting scores for the Basic CapsNet do outperform the mean scores for the other models, but more runs are needed to give certainty about these results.

8 Discussion

Several approaches could have been helpful in increasing model performance. Creating a model which determines if there are boats in an image has been shown to be useful in eliminating false positives. This model can be used either before or after the segmentation model, resulting in rejecting or not performing segmentation respectively.

Another well known approach to increase model performance is by increasing the data set by means of data augmentation, which means creating rotated, shifted and resized copies of the data set. This was not done because the original data set itself includes translated copies.

The choice of the set of variable hyper-parameters was made mostly based on previous scientific work, but this set could have been enlarged to make sure that the true hyper-parameter space was searched more exhaustively. Given more computational power, a greater space could have been explored.

Lastly, it would have been helpful to do post-processing to fit a predicted object into a rectangular shape, therefore creating a better match with a ground-truth object, increasing the intersection-over-union score for that particular predicted object.

References

1. Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. *CoRR*, abs/1710.09829, 2017.
2. Bang Liu, Yan Liu, and Kai Zhou. Image classification for dogs and cats.
3. Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.
4. Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
5. Alberto Bietti and Julien Mairal. Invariance and stability of deep convolutional representations. In *Advances in Neural Information Processing Systems*, pages 6210–6220, 2017.
6. Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with EM routing. In *International Conference on Learning Representations*, 2018.
7. Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 2019.
8. Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.

9. Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
10. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.
11. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.
12. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
13. Max Pechyonkin. Understanding hintons capsule networks. part i: Intuition. https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b.
14. Kaggle. Airbus Ship Detection Image Data Set. https://www.kaggle.com/c/airbus-ship-detection/, 2017.
15. Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.
16. Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.
17. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.
18. Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *International Conference on Artificial Neural Networks*, pages 44–51. Springer, 2011.
19. Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
20. Rodney LaLonde and Ulas Bagci. Capsules for object segmentation. *arXiv preprint arXiv:1804.04241*, 2018.