National and Kapodistrian University of Athens
Physics Departments
Department of Electronics, Computers, Telecommunications and Control

Ship Detection in Satellite Imagery Implemented on Convolutional Neural Networks

Alexandros Louropoulos

AM: 2017512

Supervisor

Dionysios Reisis
Associate Professor

Reviewers

Dionysios Reisis, Associate Professor
Anna Tzanakaki, Assistant Professor
Emmanouil X. Tsilis, Assistant Professor

Athens, September 2019

Περίληψη

Η ανάγκη για γρήγορο εντοπισμό των πλοίων σε δορυφορικές φωτογραφίες γίνεται ολοένα και πιο σημαντική για διάφορους κοινωνικούς καθώς και τεχνικούς λόγους. Ο εντοπισμός αντικειμένων σε εικόνες αποτελεί ένα γνωσιακό πεδίο της ψηφιακής επεξεργασίας σήματος στο οποίο οι τεχνικές μηχανικής μάθησης αποδίδουν σε μεγάλο βαθμό. Έτσι λοιπόν, στην παρούσα έρευνα εφαρμόζονται οι τεχνικές αυτές στην επίλυση του συγκεκριμένου προβλήματος. Συγκεκριμένα, αναπτύχθηκαν τεχνικές βαθιάς μάθησης με αρχικό σκοπό την κατηγοριοποίηση του αντικειμένου αποτελούμενα από συνελικτικά νευρωνικά δίκτυα τα οποία αναλαμβάνουν να αναλύσουν μια δοσμένη εικόνα εισόδου και να αποφανθούν για το αν η εν λόγω εικόνα περιέχει η όχι ένα πλοίο με πολύ μεγάλη ακρίβεια.

Στη συνέχεια, για την ολοκλήρωση του σκοπού (εντοπισμός πλοίου σε εικόνα), η τεχνική του συρόμενου παραθύρου αναλύθηκε στην οποία μια μεγάλη εικόνα χωρίζεται σε πολλά μικρά τμήματα τα οποία περνάνε από το νευρωνικό δίκτυο. Αν το νευρωνικό δίκτυο αποφανθεί πως στο τμήμα αυτό περιέχεται ένα πλοίο, τότε αφού είναι γνωστό σε ποιο τμήμα ανήκει, είναι γνωστή και η θέση του πλοίου. Όμως η παραπάνω τεχνική είχε ως μεγάλο μειονέκτημα τον μεγάλο χρόνο εκτέλεσης. Για να μειωθεί αυτός ο χρόνος, χρησιμοποιήθηκε η τεχνική YOLO, η οποία αποτελεί ένα συνελικτικό νευρωνικό δίκτυο που αναλαμβάνει τον εντοπισμό και χαρακτηρισμό ενός αντικειμένου. Το νευρωνικό δίκτυο αυτό καταφέρνει να επεξεργαστεί την αρχική εικόνα σε χρόνους τάξης μεγέθους μικρότερους από την τεχνική της ολίσθησης. Ωστόσο, οδηγεί στην μείωση της ακρίβειας της κατηγοριοποίησης.

Τελικώς, αναπτύχθηκε μια λύση, η οποία συνδυάζει τις δυο αυτές τεχνικές, χρησιμοποιώντας το δίκτυο YOLO ως αλγόριθμο που παράγει περιοχές ενδιαφέροντος και το απλό συνελικτικό δίκτυο ως αλγόριθμο για την τελική κατηγοριοποίηση. Οπότε ως τελικό αποτέλεσμα έχουμε έναν αλγόριθμο ο οποίος έχει και μεγάλη ταχύτητα εκτέλεσης αλλά και μεγάλη ακρίβεια στις προβλέψεις του.

Abstract

The need for real-time detection of ships in satellite imagery is increasing exponentially for various social and technical reasons. Object detection is a field of digital signal processing in which machine learning techniques are applied with high success. To that end, in this study these techniques are being applied in order to solve this task. In particular, deep learning techniques were used initially for the classification purpose. These are Convolutional Neural Networks (CNN) that receive an image and decide whether or not this image contains a ship and they do so with high accuracy.

Afterwards, in order to finish with the task (Detecting several ships in the image), a sliding window approach is attempted. In this approach, the whole image is split into several segments, each passing through the CNN in order to get the position of the classified ship. However, this technique posed challenges in the area of algorithm execution time which was high. In order to circumvent this challenge, the YOLO technique was employed that consist of a single Convolutional Neural Network that deals with both the localization and the detection of the object. The YOLO network managed to clearly outperform the sliding window approach execution time by a large margin but in doing so greatly decreased the prediction accuracy of the algorithm.

Finally, an end to end approach was developed, that combines these two techniques, using the YOLO network as a region proposal algorithm, by finetuning the threshold that YOLO outputs predictions and passing those proposals through the dedicated CNN for a final classification. This approach managed to implement an algorithm that achieves a very low execution time along with exceptional prediction accuracy.

Acknowledgements

I would to thank everyone that aided me in the making of this research and especially my supervisor Dionysios Reisis. Moreover, my family and partner deserve have my complete gratitude for their continuous support in all my endeavors.

Contents

1 Introduction

The vast increment of the available processing power has led to an ever-increasing usage of deep learning techniques. Deep learning techniques are used extensively in computer vision tasks such as image classification and object detection.

The goal of this thesis is the accurate and timely automatic ship detection in satellite images of a chosen dataset. To achieve this goal, a deep learning model is implemented, based on convolutional neural networks (CNN). The training and the evaluation processes were handled by the Tensorflow API [1] that allows quick and simpler definition of the many layers a CNN consists of and the acceleration of the computations were made using an NVIDIA CUDA GPU.

1.1 Problem Statement

Nowadays, shipping traffic is growing rapidly. More ships increase the chances of infractions at sea like environmental devastating ship accidents, piracy, illegal fishing, drug trafficking, and illegal cargo movement. This has compelled many organizations, from environmental protection agencies to insurance companies and national government authorities, to have a closer watch over the open seas. In recent years, there has been tremendous advance in the capabilities of remote sensing applications namely due to the increase resolution capabilities that the imaging satellites can achieve. This enables the usage of satellites for these tasks as the images taken from these satellites are more detailed and combined with the fact that the processing power of the satellites is increased, they can be used with machine learning and deep learning techniques for processing and remote sensing.

Satellite imagery comprises an abundant source of insights for many different fields, like finance, energy, agriculture and defense. This database is continuously growing, containing among others large sets of ports' and open sea images. The needs for these images' analysis are increased. By this analysis, in the case of sea satellite imagery, the localization and the detection of ships are aimed for purposes like maritime security. This includes among others traffic surveillance, protection against illegal fisheries, sea border violations and sea infractions prevention, oil discharge control and sea pollution monitoring.

1.2 Related Work

Surprisingly, the published literature for deep learning and machine learning techniques are not used extensively for the task of ship detection. Most implementation use thresholding techniques as well as algorithmic approaches. For example, [2] uses statistical methods, mathematical morphology and digital signal processing (DSP) techniques, e.g. wavelet analysis and radon transform for the ship extraction in optical images. Only recently [3] deep learning has been used for ships in satellite imagery, albeit in Synthetic Aperture Radar (SAR) images with great results. The study also extends to iceberg detection. The above indicate that the use of deep learning is highly beneficial in this task as it reduces the problem complexity significantly and can be extended to other areas. It is expected as deep learning becomes even more accessible, more CNN based approaches will emerge for ship detection.

1.3 Document outline

The following sections are organized as follows. Section 2 aims to give a baseline theory understanding of the technologies and methodologies that has been followed in the ship detection algorithm implementation. Section 3 defines how these methodologies were implemented and the data manipulation that has been performed. Section 4 describes the experimental results of the implemented algorithms and approaches. Section 5 summarizes the conclusions of this study and finally Section 6 proposes some future improvements that this study would benefit from.

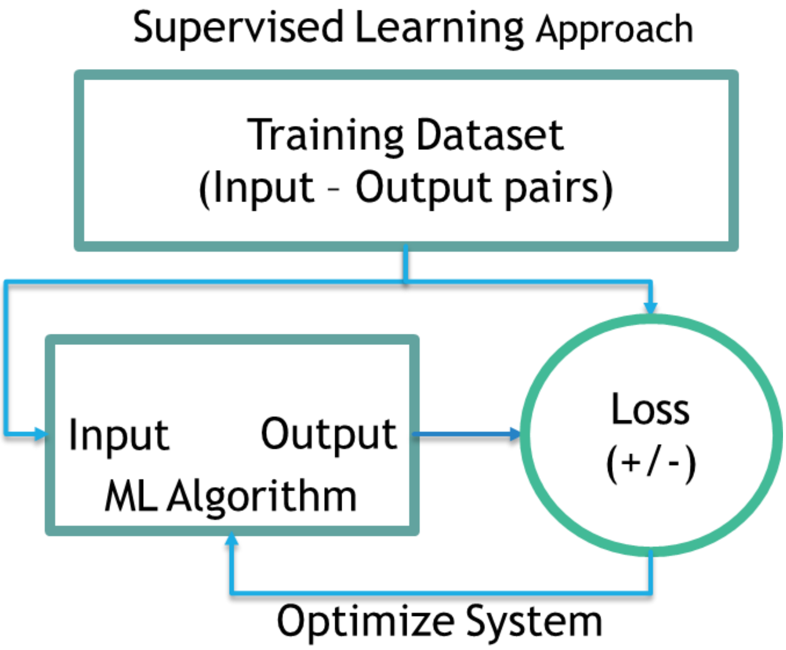
2 Background

2.1 Machine Learning

Machine learning is the art of enabling a computer system to make decisions without being explicitly programmed. In order to achieve this, a ML algorithm must be able to read data, learn from them and then make decisions based on these data about the world. Therefore, a ML algorithm will eventually create a model for a given input-output pair.

Machine learning can be categorized into the following approaches. a) Supervised learning, in which we present the ML algorithm with labeled data, i.e. input-output pairs and we follow a training process in these data. Finally, after training, it is ready to predict a similar but unknown input to an output. b) Unsupervised learning, which works on unlabeled data and has the goal to identify commonalities in the data and c) Reinforcement learning, which aims to create software agents that manage to take the best action in a task based on a given reward.

In the context of this thesis, the supervised learning approach has been explored, as the data are labeled and are a prime candidate for this kind of solution. Supervised learning is all about getting the right amount of labeled data and feeding these data through a machine learning algorithm that predicts the given label. Afterwards, we score the success or failure of the predictive model with a given loss function (how correct the prediction is) and optimize the algorithm until it produces the correct output. Finally, depending on the validity of the data, we can now observe that the ML algorithm produces the correct prediction for unknown inputs.



Machine learning algorithms can cover a variety of cases. As far as this study is concerned, the usage of these algorithms has covered the need of classification and localization for computer vision purposes.

2.1.1 Classification

A model is benchmarked and scored by its ability to map the input features to the output label, i.e. how correctly the model predicts a class (class being the possible values of the label, e.g. Ship, No-Ship). To that end we employ a number of metrics that are used for this benchmark. The below example shows the truth table of the possible prediction outcomes (also called a confusion matrix) for the Ship Class

$$TP = True\ Positive = The\ model\ predicted\ a\ Ship\ and\ it\ \boldsymbol{was\ actually}\ a\ Ship$$

$$FP = The\ model\ predicted\ a\ Ship\ and\ it\ \boldsymbol{was\ NOT}\ a\ Ship$$

$$TN = The\ model\ predicted\ a\ No - Ship\ and\ it\ \boldsymbol{was\ actually}\ a\ No - Ship$$

$$FN = The\ model\ predicted\ a\ No - Ship\ and\ it\ \boldsymbol{was\ actually}\ a\ Ship$$

2.1.1.1 Accuracy

Accuracy is the most basic metric in terms of evaluating classification algorithms. Informally, it can be defined as:

$$Accuracy = \frac{Number\ of\ Correct\ Predicitons}{Total\ Number\ of\ Predictons}$$

Formally it is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

However, accuracy is a good metric only if there are equal number of samples belonging to each class. In one case of the study, the dataset's ratio of Ship to No-Ship is $1/_4$ so a model that always predict a No-Ship will have an accuracy of around 75%. It is clear though that this model is not optimal and the metric does not represent a good measurement for model scoring thus it will not be used for benchmarking.

2.1.1.2 Recall

Recall measures the ability of a model to predict all the relevant classes, i.e. how many detractors the model predicts, out of all possible detractors. Informally, it can be defined as:

$$Recall = \frac{Correct\ Predictions}{Expected\ Predictions}$$

While the formal definition is:

$$Recall = \frac{TP}{TP + FN}$$

Ideally, Recall should be approaching one as the number of false predictions for this class approaches zero. Recall is a good benchmark that works in both balanced and unbalanced label distributions and thus will be used for the benchmark of the developed models.

2.1.1.3 Precision

Precision measures the ability of model to correctly predict the corresponding class. Informally, with this metric we care about how many predictions where a real Ship out of all the predicted Ships:

$$Precision = \frac{Correct\ Predictions}{Total\ Predictions}$$

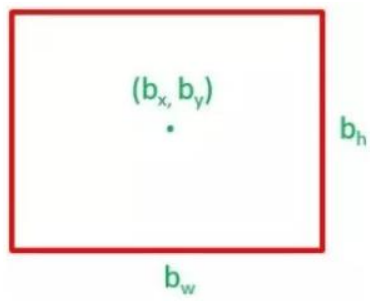
Formally, it is defined as:

$$Precision = \frac{TP}{TP + FP}$$

Precision is a valid benchmarking metric that will be used extensively.

2.1.2 Localization

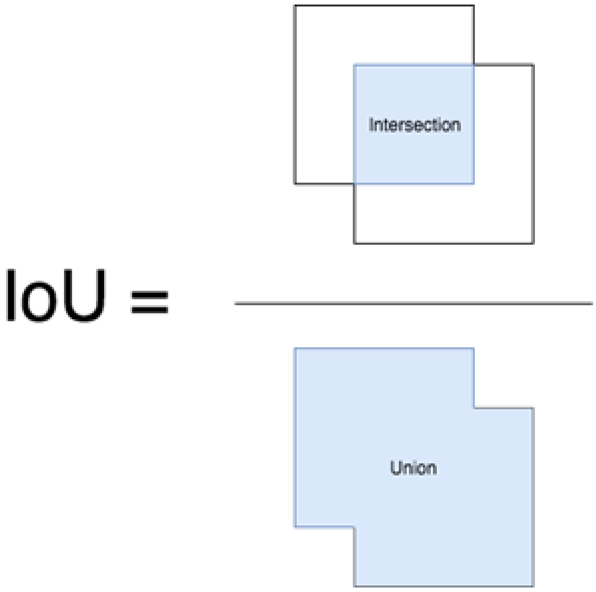
Object localization is done by defining a bounding box to mark the object along with the confidence variable P^c and the probability for the image class. This bounding box is usually is usually represented by the center (b^x, b^y), the rectangle height (b^h) and the rectangle width (b^w).



By providing this information a model is capable of predicting more information by giving a more detailed view of the image content. Consequently, by adding more points in the image a greater insight of its content can be defined. In object detection, the main objective is to improve the area of overlap between the predicted bounding box of the object and the ground truth bounding box as well as the average precision of the model, topics that will be analyzed later.

2.1.2.1 Intersection Over Union

Speaking of bounding boxes, a significant concept that involves them is Intersection over Union (IoU). IOU evaluates the overlap between two boundaries. In particular, it is used to measure how much a predicted bounding box overlaps with the ground truth one. By applying the IoU it can be decided if a detection is valid, a true positive, or false positive. IoU is given by the overlapping area between the predicted bounding box and the ground truth bounding box divided by the area of union between them:



2.1.2.2 Confidence threshold

In addition, another factor that should be taken into consideration is the confidence that the model reports for every detection. So, setting a confidence threshold is the way to specify the minimum acceptable probability that can characterize a prediction. By varying the confidence threshold, it can change whether a predicted box is considered positive or negative. Basically, all predictions, bounding boxes and classes, above the threshold are considered positive and all below it are negative.

It is highly important to note that there is an inverse relationship between precision and recall and both are intimately related to confidence threshold. Recall can be increased trivially by predicting an object everywhere regardless of if it really exists or not. However, the precision will dramatically drop. At the same time, sticking to super-high confidence detections to maximize precision, recall might suffer. So "confidence" serves the purpose of ranking the predictions and observe the relationship between the precision and recall. In this way, the model quality can be evaluated.

The Precision-Recall curve is a good way to evaluate the performance of an object detector as the confidence is changed by plotting a curve for each object class. An object detector of a particular class is considered good if its precision stays high as recall increases, meaning that different values in confidence threshold will still end up to high precision and recall.

2.1.2.3 Mean Average Precision

Aiming to evaluate the model in a model agnostic way the mean Average Precision comes in. The Mean Average Precision is a term which has different definitions. This metric is commonly used in the domains of Information Retrieval and Object Detection. Both these domains have different ways of calculating mAP.

To calculate the AP, for a specific class, the area under the precision-recall curve is found. In practice AP is the precision averaged across all recall values between 0 and 1.

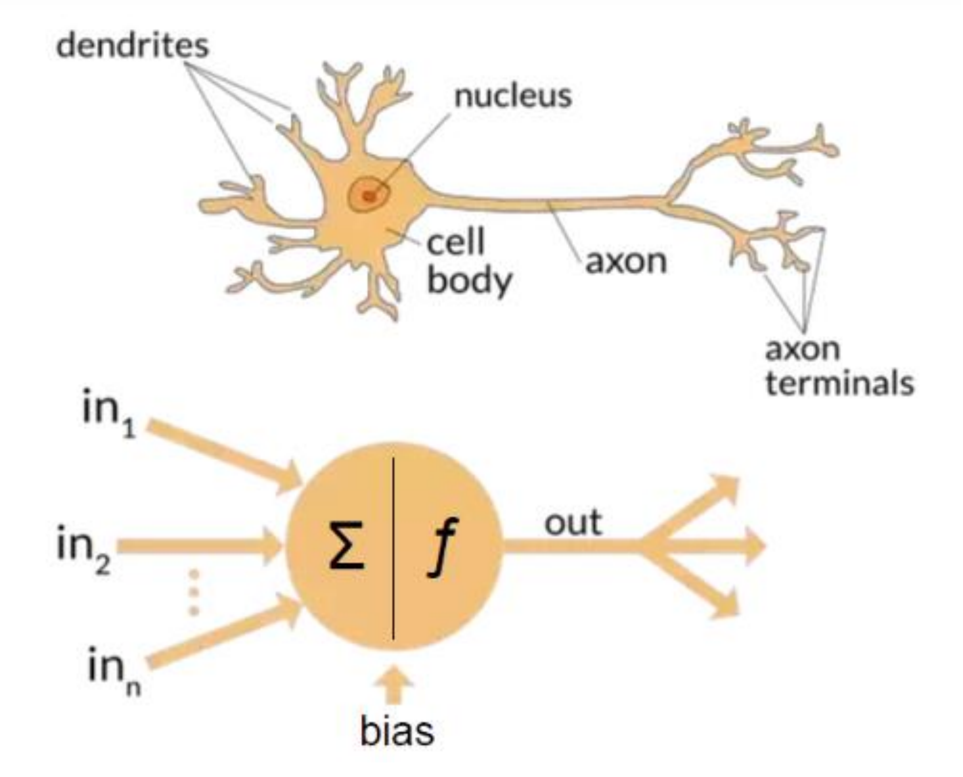
$$AP = \int_0^1 p(r)dr$$

mAP (mean average precision) is the average of AP. In some context, the AP is computed for each class and then the average of them is found.

Depending on how the classes are distributed in the training data, the Average Precision values might vary from very high for some classes, which had good training data, to very low, for classes with not so good or less data. So, mAP may be moderate, but the model might be really good for certain classes and really bad for other.

2.2 Artificial Neural Networks

Another algorithmic approach from the early machine-learning crowd, artificial neural networks, came and mostly went over the decades. Neural networks have been developed as an attempt to simulate the highly connected biological system found in the brain through the use of computer hardware and/or software. In the brain, a neuron receives input from many different sources. It integrates all of these inputs and "fires" (sending a pulse down the nerve to other connected neurons) if the result is greater than a set threshold. In the same way, a neural network has nodes (the equivalent of a neuron) that are interconnected and receive input from other nodes. Each node sums or integrates its inputs and then uses a linear or nonlinear transfer function to determine if the node should "fire".



A neural network can be arranged in several different ways. For example, it can have one or more layers of nodes, it can be fully connected (where every node is connected to every other node), or it can be partially connected. Also, it can have feed-forward processing (where processing only travels one direction), or it can have feedback processing (where processing travels both ways).

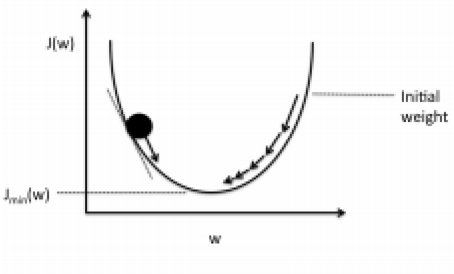
Another important aspect of neural networks is their ability to "learn" based on input patterns. A neural network can be trained in either a supervised or unsupervised mode. In the supervised mode, the net is trained by presenting it with an input and giving it the desired output. The error between the output of the net and the desired output is then propagated backward through the net, adjusting the weights of the inputs of all the nodes in such a way that the desired output is achieved. This is repeated for many input sets of training data and for multiple cycles per training set. Once the net has converged (i.e., the weights change very little for additional training sets or cycles), it can be tested with prospective data. This kind of training paradigm is very useful for finding patterns out of a known collection of patterns. Unsupervised training is similar to supervised training, but instead of providing the net with the desired output, it is free to find its own output. This type of training can be very useful for finding patterns in data where there is no known set of existing patterns. The main advantage of a neural network is the ability to solve a problem that can be represented by some sort of training data without needing an expert. However, if the training data are not complete, or if a problem is presented to the network that it has not been trained to solve, it may not give reliable answers.

2.2.1 Deep Learning

Deep learning can be thought of as a specific manifestation of ANN, with the discriminating factor being the number of layers used in the network. Models which have many hidden layers (more colloquially known as deep networks) can extract features from data at various levels of abstraction through many layers of affine transformations and non-linear functions. The success of deep neural networks (DNN) has come at the confluence of many trends: the widespread collection of various types of data, greater amounts of labelling for that data, falling hardware prices allowing for large-scale distributed computing, and more efficient network architectures just to name a few. Deep CNN have been successfully applied to many non-trivial problems; for example, aiding in the win of a computer against a world-champion at the ancient Chinese game of Go and even recognizing the onset of blindness through retinal images of patients with diabetes. DNN have also been applied to problems which are solved intuitively by humans but confound computers. Therefore, it is natural that such methods have played a major role in the development of automatic object detection systems over the past few years.

2.2.2 Backpropagation

In short, backpropagation works like an iterative cycle in which input is passed forward through the layers of the neural network and transformed via linear and non-linear computations. Once the input has reached the last layer of the network, the prediction is then compared against the ground truth label. By comparing the prediction with the ground truth, an error is calculated by a loss function. The computed loss is then used to update the weights of multiple layers of the network in order to minimize the total loss. To do so, backpropagation uses the chain rule to compute the partial derivative (the direction of the error in relation to the loss-function), given that the derivative depends on the functions of the previous layers.

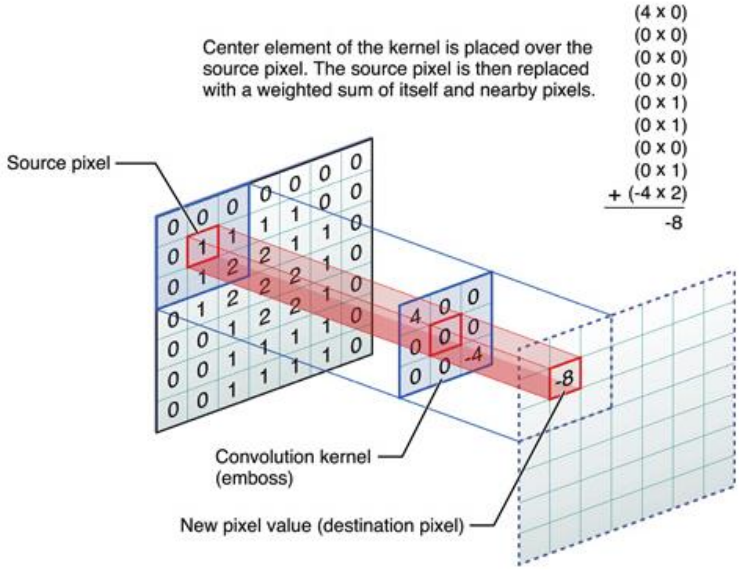


Schematic of gradient descent.

Traditionally, the fine-tuning of the weights and bias have been computed using an optimization algorithm called Stochastic Gradient Descent (SGD). SGD iteratively estimates the best direction of the weights and biases using a subset of the whole dataset to minimize the loss function, hence an incremental improvement. SGD is different from traditional gradient descent in that the weights and biases are updated after analyzing a data subset, known as batch size, which is much less computing intensive than computing the gradient for every data point. Since SGD, several alternatives have been proposed - most notably are the Adaptative Learning Methods such as ADAM. Without going into too many details, ADAM [4] works by computing the momentum of gradient descent by taking the previous weight updates into account. ADAM has proven to converge faster than SGD. To avoid changing the direction of the weights and biases too much at each batch, also known as over-fitting, a learning rate is introduced to decrease the direction of the weights by a specific factor. Careful tuning of this parameter is required for optimal training. Other techniques such as batch normalization can be used to ensure that small changes early in the network do not amplify deeper in the network.

2.3 Convolutional Neural Networks

One of the challenges with training a neural network is the number of parameters (mainly weights) that needs to be tuned. The deeper the network (number of layers) and the larger the input size, the more parameters need to be trained. In a fully connected ANN, every neuron in each layer is fully connected with every other neuron in the previous layer. Since every connection includes a weight, the number of trainable parameters increases significantly by increasing the number of initial inputs and network layers. LeNet, and other CNNs, are similar to the perceptron architecture developed by Rosenblatt in that every neuron does not fully connect to the neurons in the previous layer. Instead, a CNN is built on layers of kernels, also known as filters, which are weight matrices that are applied across the inputs of the previous layers.

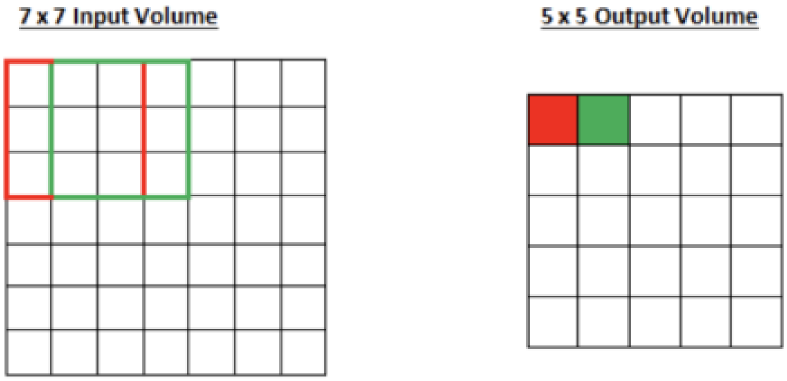

Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)

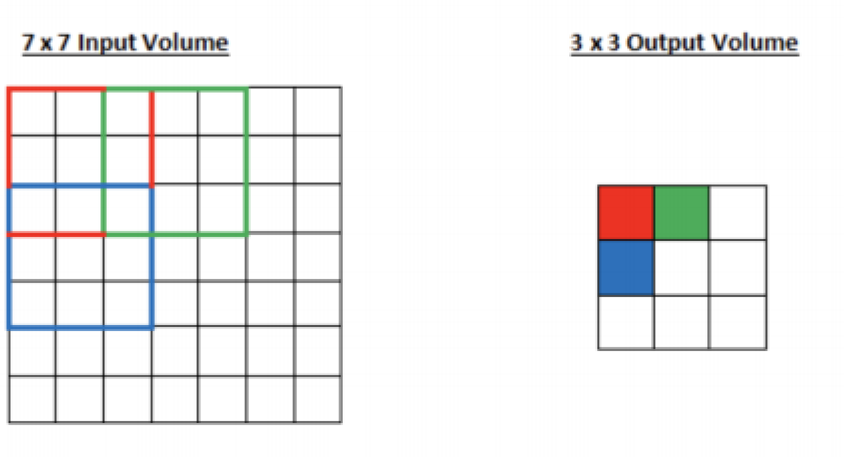
The architecture of CNNs drastically reduces the number of trainable parameters when compared to a fully connected network. As seen in the figure the weights in the kernel (also known as a filter) are applied to a specific region of the inputs space and producing a single output. By updating the weights inside the kernel through backpropagation, the kernel learns to detect certain general features from the input vector. This ability is particularly useful for image data as multiple filters learn to detect different features of various complexity (lines, objects, etc), which can be used across the entire image. Using a filter across an image is referred to as a sliding operation. The CNN characteristic of using a set of weights across an entire image is commonly known as parameter sharing. After applying a filter across the entire image, the output is referred to as a feature map. Usually, as the inputs move deeper into the CNN, the number of feature maps increase while the resolution size decrease. This happens as multiple filters are applied at every layer and the use of max-pooling, stride or no padding decreases the input size. The reason behind the dimensionality reduction, is straight-forward given a basic understanding of how filters and pooling works. Below is an introduction to the parameters of CNN filters including window size, stride and padding:

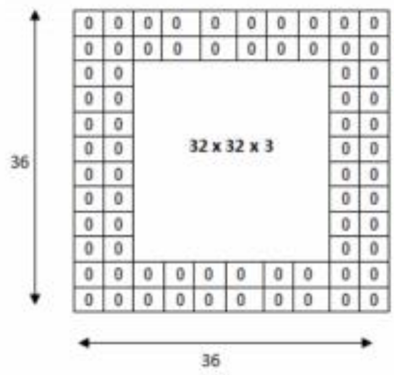
- Window size determines the size of the filters. So, in other words the number of weights.

- Stride is a parameter that determines the space between the inputs as the filter is applied.

- Padding allows the corners of an image to be analyzed by a filter as padding adds null values around the existing image







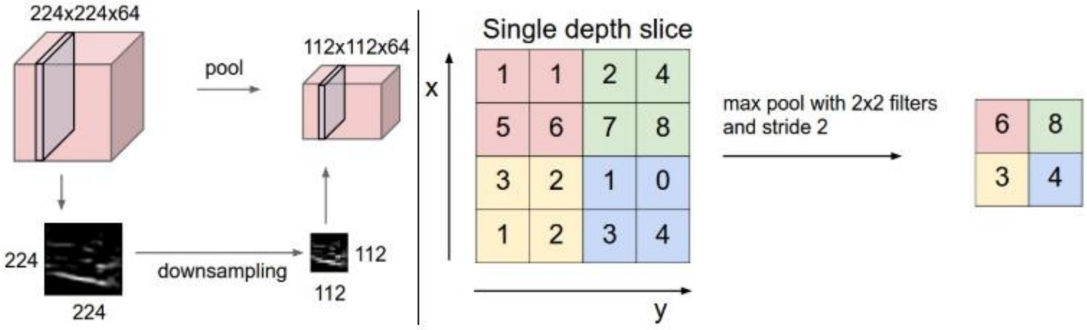
The input volume is 32 x 32 x 3. If we imagine two borders of zeros around the volume, this gives us a 36 x 36 x 3 volume. Then, when we apply our conv layer with our three 5 x 5 x 3 filters and a stride of 1, then we will also get a 32 x 32 x3 output volume.

Based on the values of the input size, stride and padding, it is possible to calculate the output size. The formula is as follows:

$$OutputSize = \frac{Input\ size - Kernel\ Size + 2 * Padding}{Stride}$$

2.3.1 Pooling

Instead of activating the feature map with convolution filters it is also possible to use a technique called pooling. Unlike filters, pooling does not rely on any weights to be tuned. Max pooling, for example, simply outputs the maximum input value of the window size as illustrated in the figure below. Whereas, another type of pooling called average pooling outputs the average value of the window size. Common to both is that the operation of pooling reduces the input size and that no parameter tuning is required.

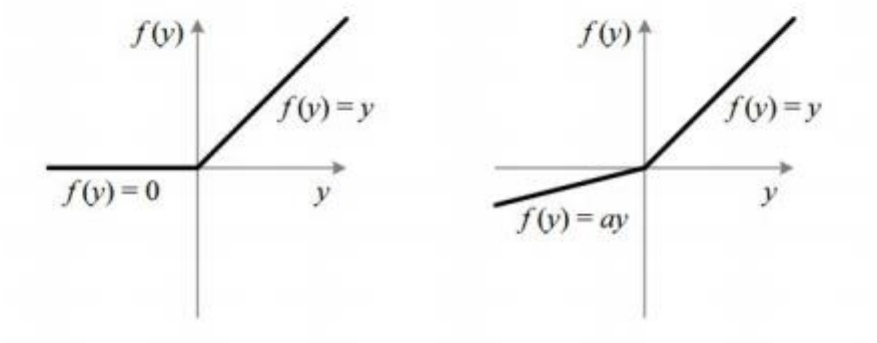


1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

2.3.2 Activation function

For CNNs, the activation function called ReLu and Leaking ReLu are often used as they provide the best performance to accuracy ratio [5] and therefore should also be briefly explained. ReLU is an example of a simple activation function; In short, it takes the input value and outputs zero if the input is zero or negative or the input if the value is positive. Leaking ReLu (on the right) is a variant of ReLU that avoids that the output is zero for inputs less than zero. The reason why it can be important to avoid zero as an output is because the value zero can subsequently turn off the next layers.



2.3.3 Dropout

The fact that the model will ignore parts of the network during training when the output value is zero can be used to reduce over-fitting. The technique, known as dropout [6], aims to ensure that certain parts of the network is not over-used and thereby causing over-fitting to the training data. It works by randomly turning off neurons during the training process (setting the output to zero), which prevents the model from over-relying on certain patterns during previous training and instead learn to identify new features that hopefully generalize better.

2.3.4 Fully Connected Layer (FC)

Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. (When activations of all nodes in one layer goes to each and every node in the next layer. When all the nodes in the Lth layer connect to all the nodes in the (L+1)th layer we call these layers fully connected layers.)

Their activations can hence be computed with a matrix multiplication followed by a bias offset. Adding a fully connected layer is a cheap way of learning non-linear combinations of these features. Most of the features from convolutional and pooling layers may be good for the classification task, but combinations of those features might be even better.

In this layer, where the weight and bias are same as the normal neural network, cost is used to compute the loss function, and gradient descent to optimize the parameters and reduce the cost function.

2.3.5 Softmax function

The output from the Fully Connected Layer is then passed through the softmax function. The softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sum to one. Finally, the node of the final layer with the highest value is the predicted class.

2.4 Object Detection

Object detection enables the detection of multiple objects in an image as well as their location. Hence, this is a more complicated task compared to classification as more information needs to be estimated and localization techniques must also be employed. Rather than just identifying the presence of a single class, an object detector is detecting multiple classes and their respective location.

It is essential to explore how different object detection architectures work.

2.4.1 Sliding Window

Current detection systems repurpose classifiers to perform detection. To detect an object, these systems take a classifier for that object and evaluate it at various locations and scales in a test image. Systems like deformable parts models (DPM) use a sliding window approach where the classifier is run at evenly spaced locations over the entire image. This method extracts features for each window and feeds them to a classification network and bounding-box regression network, the latter refining the window to more tightly envelop an object. The resulting class scores and regressed bounding-boxes are then aggregated using a greedy merging algorithm to produce the final class and location predictions. As it can be easily assumed the classification network needs to be executed each time a sliding window moves and the whole process needs to be repeated for different window sizes in order to capture accurately the different sized objects. As a result, computational cost is a huge disadvantage of the sliding window algorithm. Increasing window and stride size makes it faster but in the expense of decreased accuracy. A more intelligent alternative to using a classifier over every part of an entire image is by identifying areas that are likely to contain objects. This is referred to as generating region proposals.

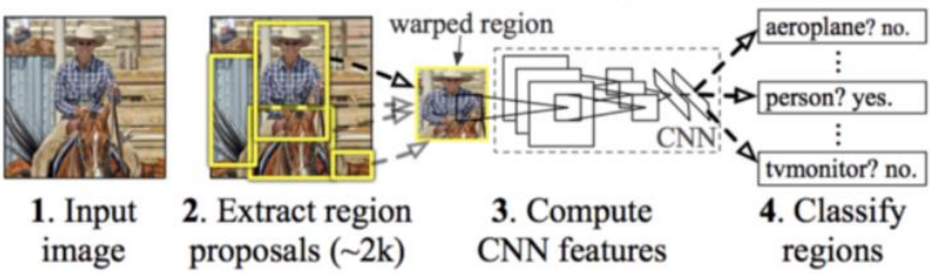
2.4.2 CNN Based Approaches

2.4.2.1 R-CNN

To bypass the problem of selecting a huge number of regions, [7] proposed a model that combines an independent region proposal algorithm and a CNN, which suggests object-containing regions and compresses them into a fixed length feature vector respectively. Each feature vector is then classified by class-specific support vector machines (SVM) and the set of classified region proposals reduced using non-maximum suppression (NMS). More precisely, the model uses the selective search algorithm to extract just 2000 regions from the image, the so-called region proposals. These 2000 candidate region proposals are warped into a square and fed into a convolutional neural network that produces a 4096-dimensional feature vector as output. Then, the extracted features are fed into the SVM to classify the presence of the object within that candidate region proposal. In addition to predicting the presence of an object within the region proposals, the algorithm also predicts four values which are offset values to increase the precision of the bounding box. Despite significantly reducing the number of regions that need to be fed into the model R-CNN comes with the following problems:

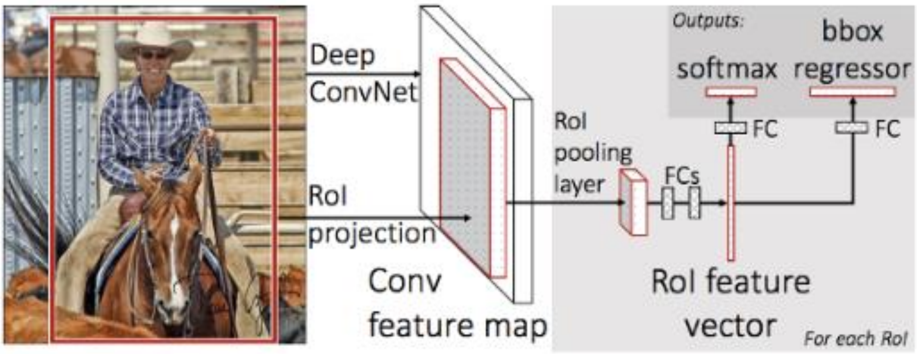
- The 2000 region proposals per image results in low performance with high training time.

- Its FPS does not allow it to be implemented in real time applications.
- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.



2.4.2.2 Fast R-CNN

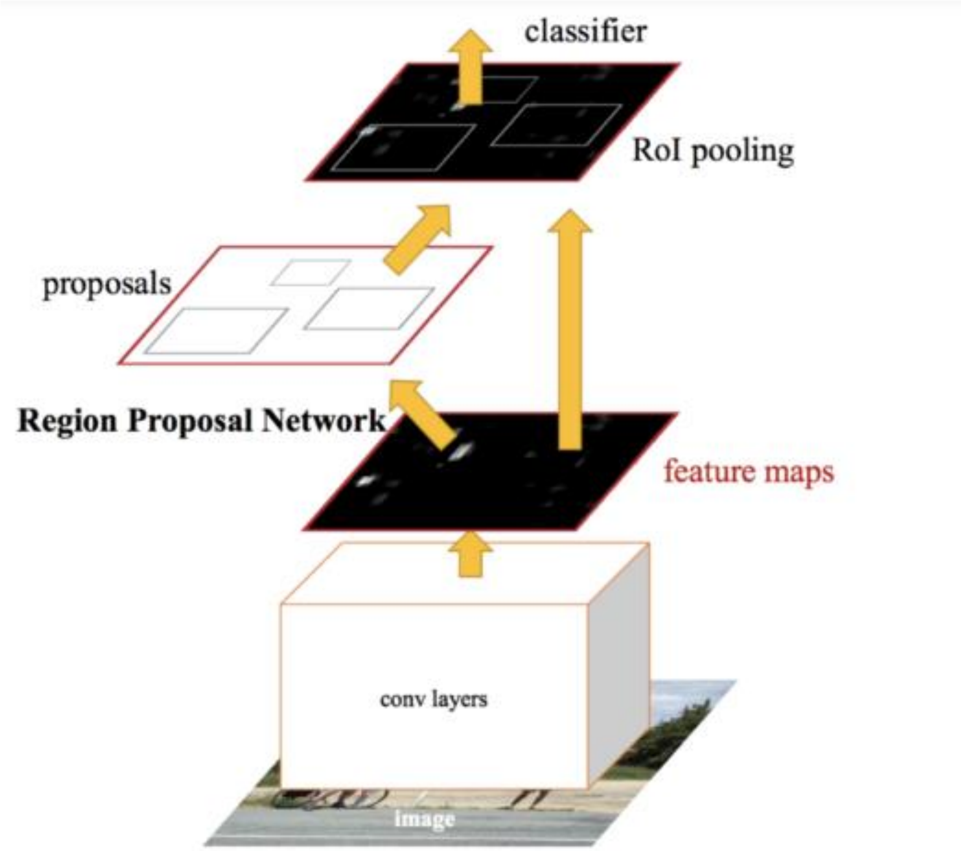
Fast R-CNN [8] retained many of the core notions of R-CNN but introduced several refinements. One such was the Region-of-Interest (ROI) pooling layer. With Fast R-CNN, the entire image is first processed by a CNN (which we will hereafter refer to as a feature extractor in this context), producing a single set of features maps for the whole image. As with R-CNN, region proposals are generated externally. On the contrary, each spatial region is then projected onto the feature maps, and the associated volume is pooled to standard dimensions. The proceeding fully connected layers take as input these features and output softmax class confidence scores and bounding-box regression offsets.



2.4.2.3 Faster R-CNN

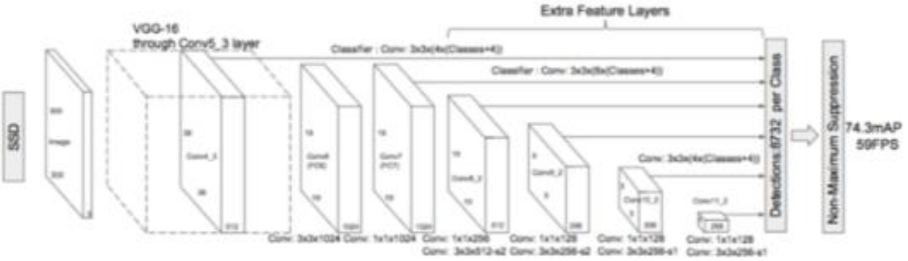
These two algorithms (R-CNN & Fast R-CNN) uses selective search to find out the region proposals. Selective search is a slow and time-consuming process affecting the performance of the network. Therefore, [9] designed an object detection algorithm that eliminates the selective search algorithm and lets the network learn the region proposals.

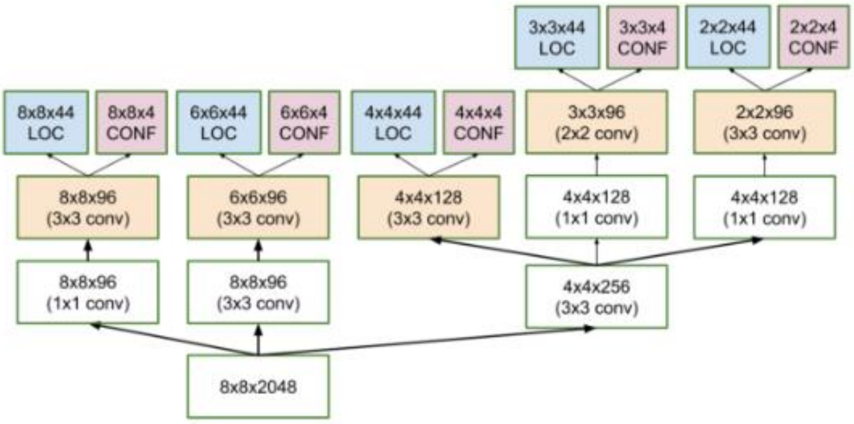
Like Fast R-CNN, in Faster R-CNN the image is provided as an input to a convolutional network which provides a convolutional feature map. Instead of using selective search algorithm on the feature map to identify the region proposals, a separate network, called Region Proposal Network (RPN), is used to predict the region proposals. In this way, a single set of feature maps is generated from the input image, which is then passed to two sibling networks, the RPN which generates region proposals, and a Fast R-CNN network for classification and regression of these ROI.



2.4.2.4 SSD

SSD [10], also known as Single Shot MultiBox Detector, is inspired by the successfully CNN classifier architecture known as VGG-16, but rather than the fully connected layers in the end for object classification, the architecture has been revised to detect class-agnostic boundary boxes based on feature maps from different layers in the network. This approach is known as a single shot approach. The illustrations below show how boundary boxes are detected from multiple layers in the network.





To optimize the predictions of the boundary boxes, the SSD loss function computes both confidence and location loss. The confidence loss measures how confident the network is at predicting the specific class. Categorical cross-entropy is used to compute this loss. Location Loss measures how far away the predicted bounding boxes are from the ground truth from the training set. L2-Norm is used to compute the location loss. SSD has become known as one of the most accurate approaches to object detection.

2.4.2.5 YOLO (You Only Look Once)

While the algorithms presented have achieved efficient and accurate models with high mAP their Frames per Second (FPS) render them lackluster for real-time detection. YOLO's [11] FPS on the other hand make it ideal for real-time applications. YOLO achieves that high detections time by using a single convolutional neural network to simultaneously predict multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance.

YOLO's benefits over the other methods of object detection are presented below:

It is extremely fast. Detection is framed as a regression problem and not as a complex pipeline. It is simply executed on a new image at test time to predict detections.

- It reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time, so it encodes contextual information about classes as well as their appearance.
- It learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected input.

YOLO divides the input image into a S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the given box contains an object and also how accurate it thinks the box is that it predicts. Formally we define confidence as Pr(Object) $*$ IOU, where IOU which stands for Intersection Over Union is simply a ratio. In the numerator the overlap of the predicted bounding box and the ground-truth bounding box is computed while in the denominator, the area of Union or simply the area encompassed by both the predicted bounding box and the ground-truth bounding box. If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the IOU between the predicted box and the ground truth. Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Each grid cell also predicts C conditional class probabilities, $Pr(Class_i \,|Object)$. These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B. At test time we multiply the conditional class probabilities and the individual box confidence predictions,

$$Pr(Class_i|Object) * Pr(Object) * IOU = Pr(Class_i) * IOU$$

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

In total, the model outputs S × S × B bounding boxes since each grid cell predicts B bounding boxes. Undoubtedly, it is essential to filter the algorithm's output down to a much smaller number of detected objects. In order to achieve that, two techniques are applied:

1. Score-thresholding: Throw away boxes that have a class with a score less than the threshold
2. Non-maximum suppression (NMS)

The steps to perform NMS:

- While there are any remaining boxes:
    - Pick the box with the largest pc and output that as a prediction

Discard any remaining box with IOU >= IOU threshold with the box output in the previous step

3 Detector Implementation

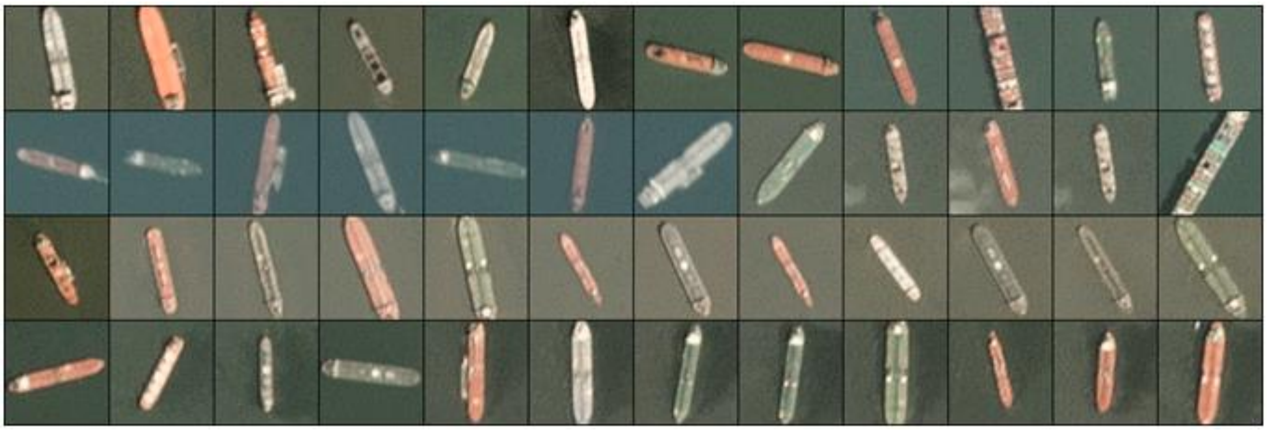
3.1 Sliding Window Approach

3.1.1 Dataset Exploration

The dataset that was chosen to feed the classifier for the sliding window approach is the “Ships in Satellite Imagery” [12] Kaggle dataset. It consists of image segments extracted from Planet satellite imagery, collected over the San Francisco Bay and San Pedro Bay areas of California. It contains 4000 80x80 RGB images in total, labeled with either “ship” or “no-ship” classification. Image chips were extracted from PlanetScope full-frame visual scene products, which are orthorectified to a 3-meter pixel size.

3.1.1.1 Class Labels

In the dataset, the 1000 out of 4000 are “ship” class images. The extraction of these images was performed in such a way that the derived ones are near centered on the body of a single ship. Ships of different sizes, orientations, and atmospheric collection conditions are included. An image may contain a bigger than its dimensions ship, as long as it is centered. Example “ship” class images are shown below:



The rest 3000 images in the dataset constitute the “no-ship” class images. A third of these are randomly sampled of different landcover features such as water, vegetation, bare earth, buildings, etc. These images do not include any portion of ship, at all. The next third are partial ship images that contain only a “definitely not” centered portion of a ship, not enough to meet the full definition of the “ship” class. The last third consists of images that have previously been mislabeled by machine learning models, typically caused by bright pixels or strong linear features. It is worth mentioned that chris-crafts are also assumed as “no-ships”. Example “no ship” class images are shown below:

3.1.2 CNN architecture

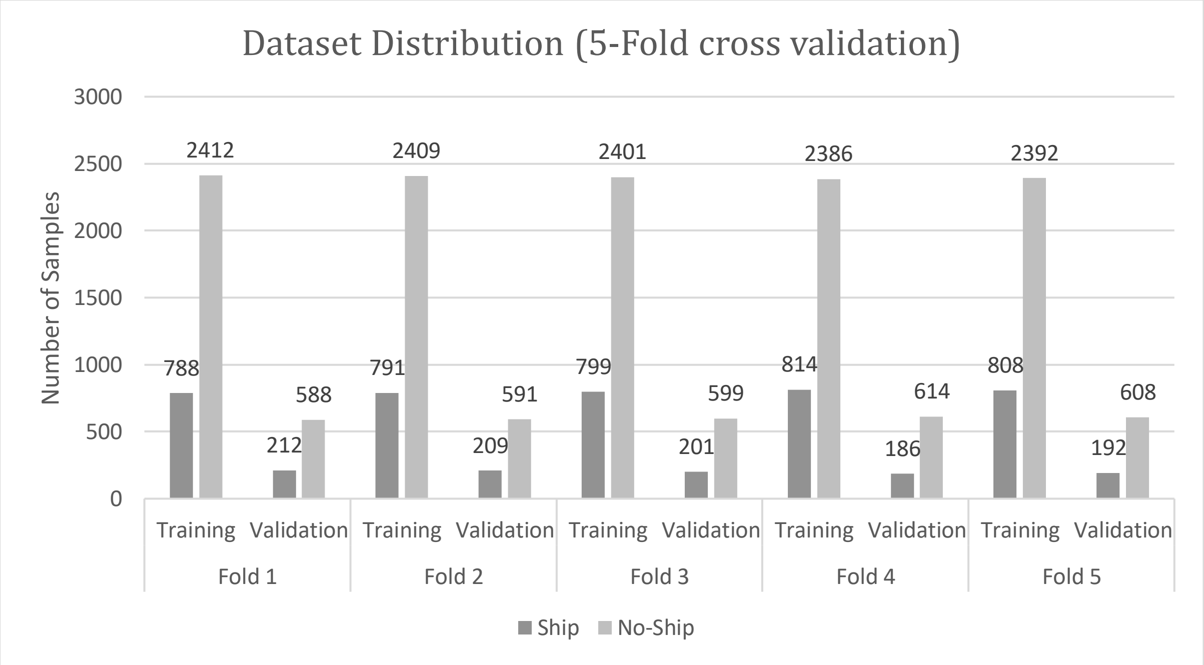
In order to collect a thorough understanding of how deep learning performs on the given objective (finding ships in satellite images), several different classifiers were trained. The classifiers mostly range in depth and complexity having various numbers of convolution layers and either one or two fully connected layers. The produced architectures are then compared against each other in terms of the general model metrics (i.e. accuracy, recall and precision), the inference time for one image as well as the number of parameters that each Neural Network has. The rationale behind benchmarking the inference time and the number of parameters falls under the fact that we assume that the chosen architecture will be required to operate in restricted environments (i.e. satellites with a limited amount of memory and processing power). Therefore, it is imperative that we choose a network that achieves the highest amount of ship detection accuracy with the least resource cost.

As far as the architecture itself, we concluded that for the given RGB input of 80x80x3 nodes, the convolutional layers should have kernels of 7x7 with 32 filters, the max pooling layers should downsample their input by half and fully connected layers of 128 nodes are enough for the binary classification. For all these layers a ReLU activation function was chosen. Finally, the final layer contains 2 nodes, each representing one class (ship and no-ship). Their output is finally routed through a softmax as well as an argmax function to generate the final classification. The network is trained for 10 epochs in batches of 32, which translates in 1000 training steps using the binary cross entropy loss function [13] and the Adam Optimizer to calculate the gradients and adjust the neural network weights.

The following figure depicts the different CNNs used with their respective number of parameters, their training and their inference time (for 800 images of the validation dataset, pipelined in batches of 32) for a GTX 960 GPU.

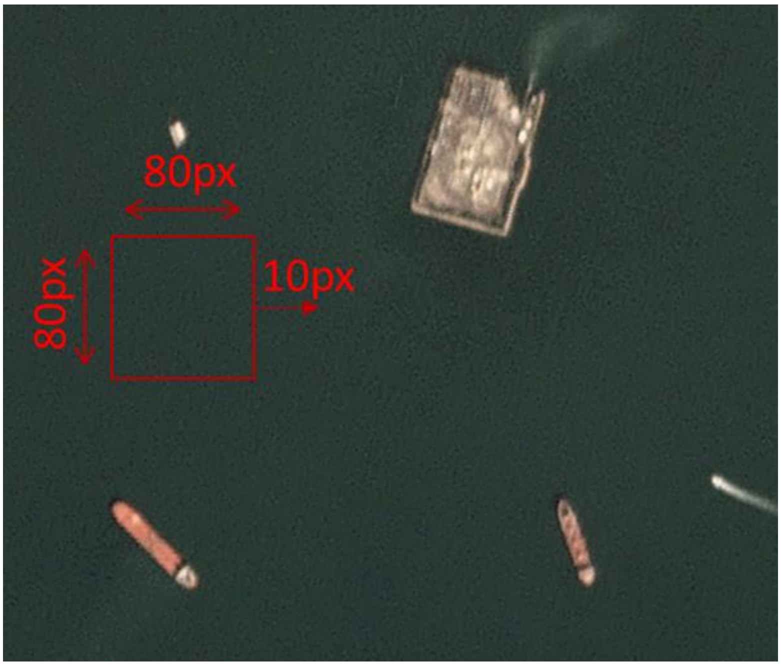
CNN Architecture	Conv Layers	Fully Connected Layers	Number of Parameters (K)	Size (MB) (32bit Float)	Training Time (mm:ss)	Inference Time (μs)
classifier_4c_2fc	4(7x7x32)	2 (128)	269	1.08	8:17	3.3
classifier_4c_1fc	4(7x7x32)	1 (128)	253	1.01	8:14	1.2
classifier_3c_2fc	3(7x7x32)	2 (128)	526	2.10	8:05	1.0
classifier_2c_2fc	2(7x7x32)	2 (128)	1705	6.82	7:19	1.0
classifier_2c_1fc	2(7x7x32)	1 (128)	1689	6.76	7:15	1.1
classifier_1c_2fc	1(7x7x32)	2 (128)	6557	26.23	5:20	1.2
classifier_1c_1fc	1(7x7x32)	1 (128)	6554	26.22	5:20	1.1

Another technique that has been used for the purposes of choosing the best split of training and evaluation datasets is the k-fold cross validation technique allows for a better data utilization as it alternates between k different sets of training and validation data and aims to maximize the accuracy. In this study, a 5-Fold cross validation has been selected and we selected the one that achieves the highest scores. Caution has been taken that each fold contains the correct class distribution as well as the same training and validation dataset distribution. The following figure shows this dataset behavior.



For the sliding window detection part, an algorithm has been developed that slices a large image scene, as seen in the following figure, in segments. These segments are 80x80 in size and to ensure that a ship will have a higher change to be placed in the center of the segment, the sliding window has a stride size of 10 pixels. After this procedure, these segments are fed through the classifier in batches where they are labeled based on the detection of ship. In the chance that a ship is detected, the position of the ship is stores and then a bounding box is drawn on the initial scene.

Finally, after all the segments have been visited the algorithm produces the scene with the new bounding boxes.



3.2 CNN-Based Approach

In the previous chapter, a variety of CNN-Based Approaches were introduced and analyzed. Most of these algorithms operate in a similar manner. However, given the fact that ship detection algorithms are executed in resource restricted environments, a fast and memory efficient algorithm must be chosen. The best candidate from the previously mentioned algorithms is the YOLO implementation which consist of only of a single CNN network with roughly a million and a half parameters that can generate both the classification and the localization of an object. Therefore, the YOLO network was selected as the algorithm that will be analyzed for this scenario as it fulfills all the requirements for the ship detection on satellites objective.

3.2.1 Dataset Exploration

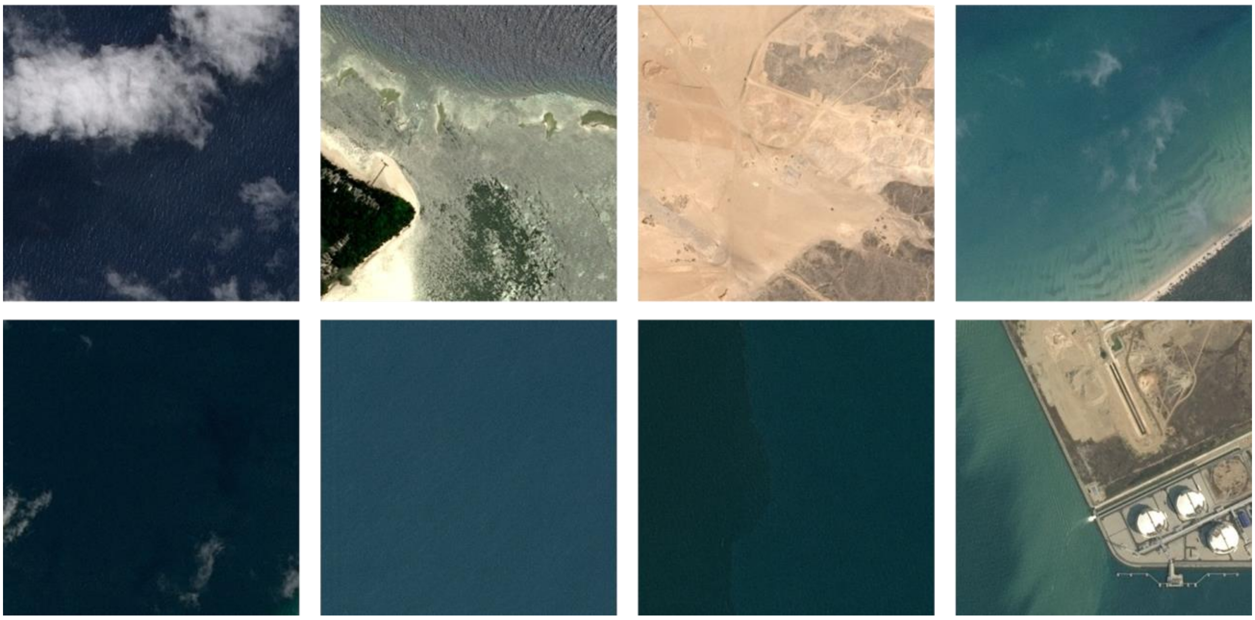
Aiming to enhance the model's performance, omitting the tidal-wave barrier false classifications, the intention was to use another dataset with an abundance of samples. Specifically, the needs for the new dataset was to contain many more "ship" class images, containing ships with even bigger selection of dimensions. Moreover, a wide variety of "no-ship" images was vital, including more edge cases like different sea scenes and most assuredly, tidal-barriers, to avoid the false positive detections, that would end up making the model more generative, even if it means producing a deeper CNN.

The dataset that was selected to be used for the training and evaluating of the implemented YOLO detector was the "Airbus Ship Detection Challenge" [14] Kaggle dataset. Airbus offers comprehensive maritime monitoring services by building a meaningful solution for wide coverage, fine details, intensive monitoring, premium reactivity and interpretation response. Combining its proprietary data with highly trained analysts, they help to support the maritime industry to increase knowledge, anticipate threats, trigger alerts, and improve efficiency at sea.

The dataset includes 132000 768x768 RGB images. The 42000 of the total images are the "ship" class images. Ships within and across those images may differ, sometimes significantly, in both shape and size and be located at various possible places like in open sea, at docks, marinas, etc. The rest 90000 images contain no ships. Moreover, the samples also contains a larger variety of sea examples like sea with waves, oil puddles, shallow waters and also challenges and obstacles in the sense of noise, clouds, tidal barriers.

However, the most significant difference between the first and the second dataset explored, apart from the number of samples each dataset contained, lies in the fact that the airbus dataset, the "ship" images are no longer ships extracted from larger ones, containing only the object of interest. Conversely, whole scenes are provided to define the "ship" class and as a result, among them, there are many that contain more than one ships.

Ship Class Images


No-Ship Images

As the images were no longer cropped, a way to determine the ships on each one of them was needed. To solve this problem an additional information file is also provided by the dataset. It supplies the ground truth, in run-length encoding (RLE) format, for the training images. Specifically, RLE representation refers to a very simple form of lossless data compression in which runs of data are stored as a single data value and count, replacing the original run. In the case of the dataset's "ship" images the .csv file contains a single line for those which depict exactly one ship and multiples lines for those which illustrate more than one ships. Each line starts with the image name followed by a list of tuples consisting of the location and the maximum number of bits which outline the ship. There is also a line for each "no-ship" image including only the image name.

The format necessary for a localization task and namely the YOLO detector to be trained is a .txt file for each .jpg "ship" image file of the training dataset, named after the image. The file should contain a line for each object of interest comprised of the class number that the object belongs to and its relative coordinated on the image. In particular, the format of each line should be the following:

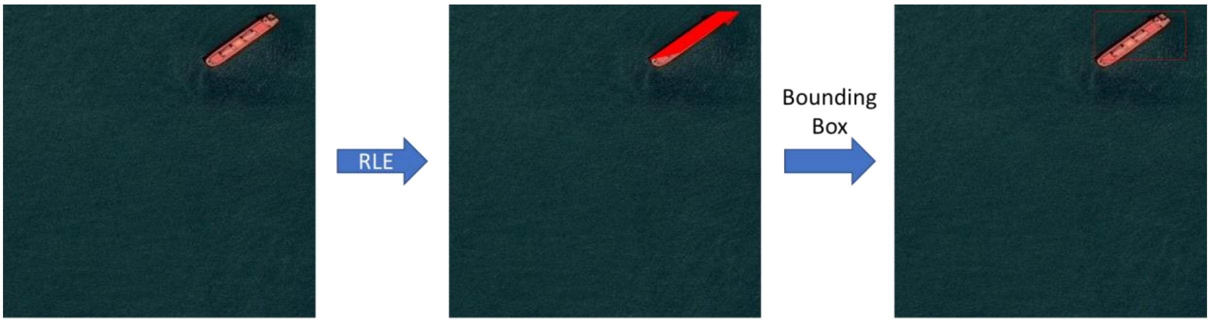
$$< object - class > < x > < y > < width > < height >$$

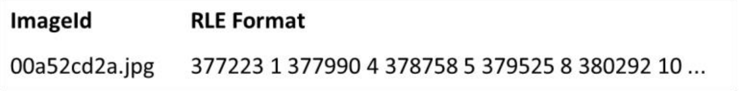
Where:

- <object-class> - an integer number from 0 to (available classes - 1)
- <x> <y> <width> <height> - float numbers relative to the image's width and height with acceptable values in the range of 0.0 and 1.0. They are calculated as follows:

$$< x > = < object_x >/< image_{width} >$$
$$< y > = < object_y >/< image_{height} >$$
$$< width > = < object_{width} >/< image_{width} >$$
$$< height > = < object_{height} >/< \ image_{height} >$$

in which <object_x>, <object_y>, <object_width>, <object_height> the rectangle's, that encloses the object, corresponding coordinates are implied. An example is shown below:





ImageId	RLE Format
00a52cd2a.jpg	377223 1 377990 4 378758 5 379525 8 380292 10 ...

Bounding Box Format
0 0.78515625 0.09895833333333333 0.2916666666666667 0.15364583333333334

Consequently, starting from RLE representation the goal was to produce the .txt file described above for each image. At first, the objects of interest on each "ship" image had to be marked with a rectangle. To do so, a python script was developed to perform the transformation between RLE and rectangle. Finally, having the image dimensions, along with the derived rectangle coordinates it was made possible to generate a .txt file with the intended YOLO compatible format, again with the help of a python script. It is worth mentioning that for every image with more than one ships a multiline file was created, whereas for each "no-ship" image an empty file is required.

3.2.2 YOLO Object Detector

The YOLO implementation was based on the official darknet source [15] that contains detailed instruction on how to build and create custom detectors. Due to memory constraints of the GTX 960 GPU (2GB) the full YOLOv3 architecture was unable to be used. Therefore, a smaller and less accurate architecture named YOLOv3-tiny was used that is around half the size of normal YOLO. The detection is done in the YOLO layers, by applying 1 x 1 detection kernels on feature maps of different sizes at different places in the network. Tiny YOLO v3 uses 2 yolo layers while its bigger brother uses 3. Both versions of YOLO v3, in total uses 9 anchor boxes, which are generated by using K-Means clustering. They are assigned in descending order of dimensions. In Tiny YOLO, the 5 biggest anchors are assigned in the first yolo layer and the other 4 in the second.

The first step of the YOLO adaptation is to refine the model and its hyper parameters. These are defined on a configuration file with the source. Most of the configuration happens in this file. We can set how many batches to load into the GPU in each iteration, how much the width and height of the input layer will be and we can also set an amount of data augmentation for darknet to apply to the training dataset like saturating or rotating images before being fed into the network. Then we define the learning rate and the rate of its decay, the optimizer method (Adam in this case). Finally, we explicitly define the convolutional, down sampling and YOLO layers.

We implemented the latest version of YOLO, YOLO v3 (the Tiny Version). The most noticeable new feature of YOLO v3 is that it makes detections in different scales. The detection is done in the YOLO layers, by applying 1 x 1 detection kernels on feature maps of different sizes at different places in the network. Tiny YOLO v3 uses 2 yolo layers while its bigger brother uses 3. Both versions of YOLO v3, in total uses 9 anchor boxes, which are generated by using K-Means clustering. They are assigned in descending order of dimensions. In Tiny YOLO, the 5 biggest anchors are assigned in the first yolo layer and the other 4 in the second.

Detections at different layers help address the issue of detecting small objects, a frequent complaint with YOLO v2. The upsampled layers concatenated with the previous layers help preserve the fine-grained features which assist in detecting small objects.

Generally, there is no need to change any of these layers except the YOLO layers and their previous convolutional layers. The convolutional filters before the YOLO layers, which depend on the classes, cords and number of masks, in each of the 2 YOLO layers owe to follow the below rule:

$$filters = (classes + 5) * 3$$

Finally, we are left with a CNN architecture that requires 52MB of memory, which roughly translates to 1.6million parameters and can infer a whole image in 8.3ms. This means that a general region proposal prediction of ships in a satellite image is 14k times faster than the sliding window approach.

3.2.3 Threshold Finetuning

Now that the YOLO detector has been trained and the first benchmark results are documented, it is shown that the detector has a low score of recall. That means that the network does a poor job of correctly detecting ships. To address this behavior, YOLO allows us to set a confidence threshold in which the algorithm will flag ships that is more or less certain, depending on this threshold. Therefore, for the final step of the YOLO detector implementation, we sought to find how the different confidence thresholds behave on the reference scene.

3.3 Ensemble Approach

As a final step, we chose to combine the speed of the YOLO network with the developed classifier in order to increase the precision as well as the performance of the implementation. To that end, we initially feed the image through the YOLO network, creating region proposals, that means regions with higher than zero, albeit low confidence that may or may not contain ships. Afterwards, since these regions are much less in quantity than the whole image segments, their coordinates are exported and then read from the python script. Finally, the segment coordinates are passed through the dedicated classifier for a higher confidence prediction and if a ship is predicted, its bounding box is drawn on the initial image.

4 Experimental Results

4.1 Sliding Window Approach

4.1.1 General Benchmark Metrics

For the sliding window approach, we benchmark the success of the model on the Precision and Recall metrics of the Ship class along with their running time. The better the recall metric, the more successful the algorithm is in detecting ships. Moreover, as the precision metric becomes higher, the algorithm's predictions are more accurate. In the following graph, the respected fold for each architecture runs are presented.

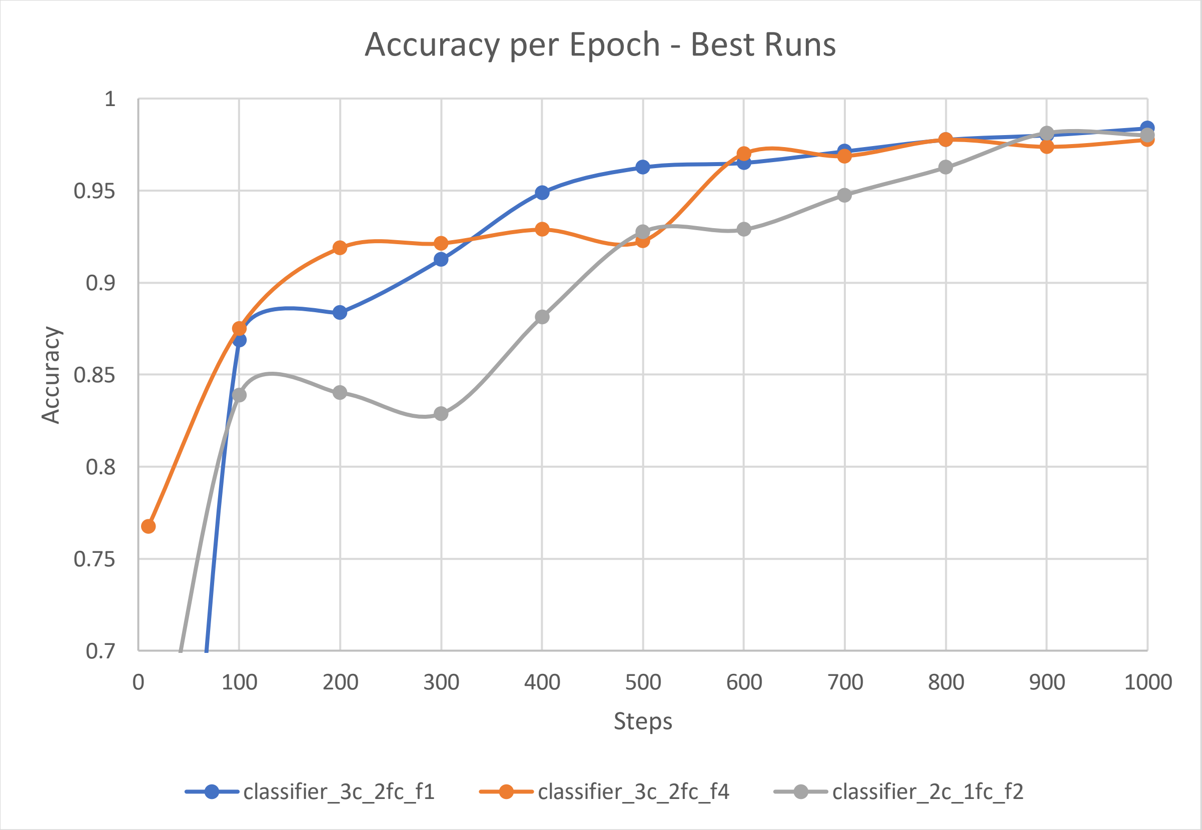
Fold	Fold 1		Fold 2		Fold 3		Fold 4		Fold 5	
Arch	4c_2fc_f1		4c_2fc_f2		4c_2fc_f3		4c_2fc_f4		4c_2fc_f5	
Label	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship
Recall	0.00	1.00	0.94	0.97	0.98	0.97	0.91	0.99	0.00	1.00
Precision	0.00	0.73	0.92	0.98	0.91	0.99	0.96	0.97	0.00	0.76
Arch	4c_1fc_f1		4c_1fc_f2		4c_1fc_f3		4c_1fc_f3		4c_1fc_f4	
Label	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship
Recall	0.00	1.00	0.94	0.92	0.00	1.00	0.00	1.00	0.00	1.00
Precision	0.00	0.73	0.81	0.98	0.00	0.75	0.00	0.77	0.00	0.76
Arch	3c_2fc_f1		3c_2fc_f2		3c_2fc_f3		3c_2fc_f4		3c_2fc_f5	
Label	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship
Recall	0.97	0.99	0.79	1.00	0.97	0.97	0.95	0.99	0.97	0.95
Precision	0.97	0.99	0.99	0.93	0.92	0.99	0.96	0.98	0.87	0.99
Arch	2c_2fc_f1		2c_2fc_f2		2c_2fc_f3		2c_2fc_f4		2c_2fc_f5	
Label	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship
Recall	0.75	0.91	0.99	0.97	0.59	0.99	0.53	0.99	0.97	0.97
Precision	0.75	0.91	0.92	0.98	0.94	0.88	0.94	0.87	0.90	0.99
Arch	2c_1fc_f1		2c_1fc_f2		2c_1fc_f3		2c_1fc_f4		2c_1fc_f5	
Label	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship
Recall	0.90	0.95	0.98	0.98	0.98	0.94	0.96	0.96	0.84	0.90
Precision	0.86	0.96	0.94	0.99	0.86	0.99	0.87	0.99	0.73	0.95
Arch	1c_2fc_f1		1c_2fc_f2		1c_2fc_f3		1c_2fc_f4		1c_2fc_f5	
Label	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship
Recall	0.88	0.95	0.98	0.96	0.93	0.99	0.96	0.95	0.94	0.95
Precision	0.86	0.96	0.90	0.99	0.96	0.98	0.86	0.99	0.86	0.98
Arch	1c_1fc_f1		1c_1fc_f2		1c_1fc_f3		1c_1fc_f4		1c_1fc_f5	
Label	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship	Ship	No-Ship
Recall	0.82	0.98	0.96	0.94	0.87	0.94	0.80	0.97	0.88	0.99
Precision	0.95	0.94	0.86	0.99	0.84	0.95	0.89	0.94	0.98	0.96

As is evident, some runs were unable to train at all. This is due to the fact that the dataset is unbalanced and in the initialization phase, a lot of the no-ship class samples were randomly fed through the neural network, making it create a bias towards this class. Consequently, it learned to always predict a no-ship giving the model an accuracy of $\frac{3}{4} = 0.75\%$

Ultimately, we want to narrow down the architecture selection to the networks that manage to correctly classify the most ships, without having too many false positives. This leaves us with the networks that have more than 95% on the Recall and Precision metric for the Ship class.

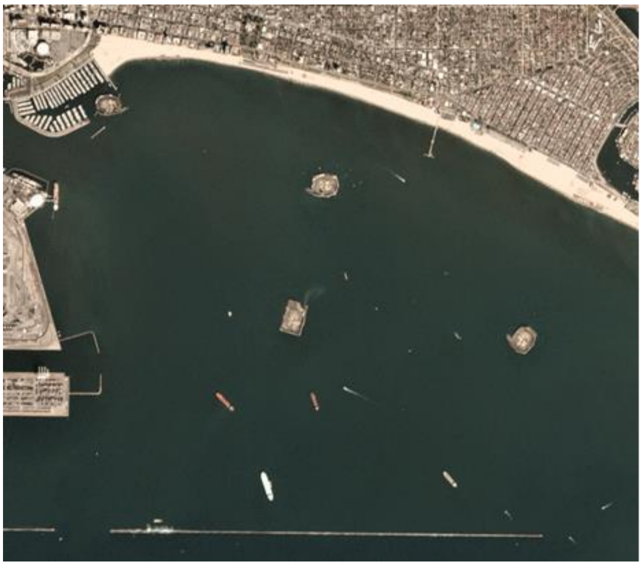
1) 3c_2fc_f1 (Recall = 97%, Precision = 97%)
2) 3c_2fc_f4 (Recall = 95%, Precision = 96%)
3) 2c_1fc_f2 (Recall = 98%, Precision = 94%)

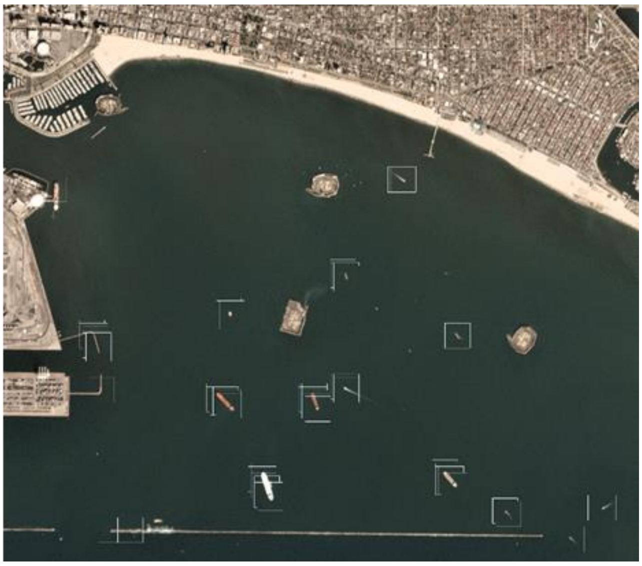
And the respective accuracy over training time:



These metrics, combined with the inference time and total memory requirement, lead us to choose architecture 3c_2fc as the final architecture for the sliding window object detector. Firstly, the architecture is beneficial due to the fact that it manages to score good metrics in two separate folds, making it more prone to generalizing. Moreover, it has an inference time of $1\mu s$ with a memory footprint of just over $2MB$ making it ideal for constrained environments.

Using this architecture, along with the sliding window approach on the reference scene image requires 2 minutes and 20 seconds on an overclocked Ryzen 2600 – GTX 960 machine. The algorithm draws bounding boxes around every image segment that it classifies as a ship. Below are the reference images provided by the dataset before and after the algorithmic implementation.





As is evident from the bounding boxes, the convolutional neural network is successful in finding the ships from the image scenes. In the first example, only one ship in the open water was unable to be detected as it was too close to the port, and the dataset had no training samples of similar situations. However, the ship that was close to the dock was properly detected, a product of having a similar sample in the training dataset.

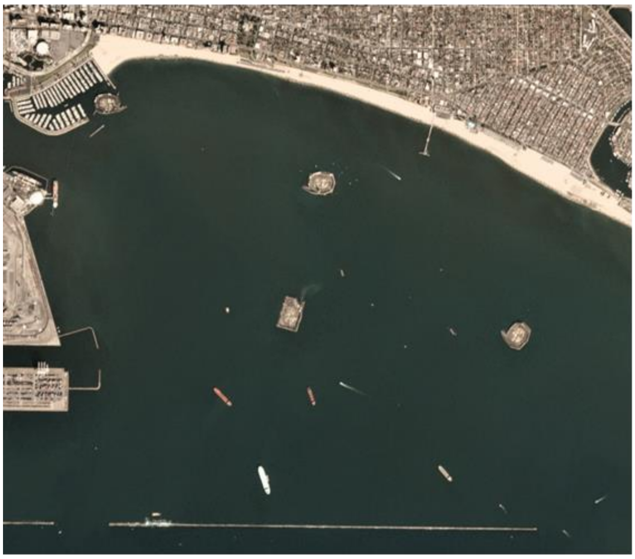
From the above images, we can draw the conclusion that the classifier does a poor job of correctly ignoring tidal waves. The reason for such behavior is the limited amount of tidal wave barriers in the dataset. No matter how many tidal wave barrier examples falls under the training or validation dataset, they are not enough for the neural network architecture to properly train on them and learn to ignore them. Having this many training examples, it would be required for the classifier to be trained for a lot more epochs. However, by doing so we encountered overfitting issues even though the dropout method was applied.

Another countermeasure that was attempted is the augmentation of the tidal barriers. The images containing tidal barriers were rotated and flipped in order to create more samples, but the situation did not improve. This finding also initially led to the conclusion that another, more complete dataset should be explored.

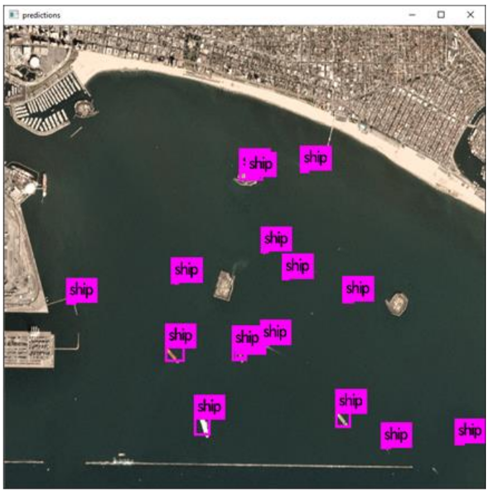
Another important finding is that the algorithm is predicting the same ships multiple times. This occurs because there is no failsafe mechanism in place to ensure that once a ship has been detected to skip it for the remainder of the sliding window steps. It turns out it is an increasingly difficult task to ignore segments based on IoU because there might be several ships close to each other, making the algorithm ignore some ships from being detected.

4.2 CNN-Approach

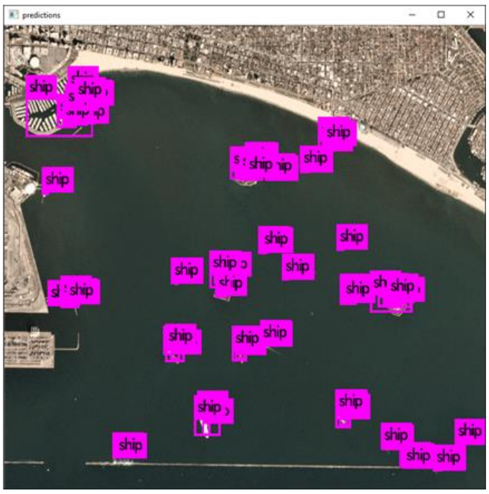
Below, two indicative examples of ship detection are cited along with the bare image. The first example shows the resulting bounding boxes using confidence threshold equal to 0.25, whilst the second one is the outcome of confidence threshold equal to 0.10. It is obvious that by using higher threshold the predictions are fewer and generally accurate, as all ships are detected just once. Moving on to the second example, with the lower threshold, the bounding boxes are increased, introducing many false positive detections, which, according to all the above, is the expected behavior. Someone could ask why a higher threshold wasn't applied but doing this would have led to loss of true positives without adequately decreasing the false negatives.



25% Threshold



10% Threshold

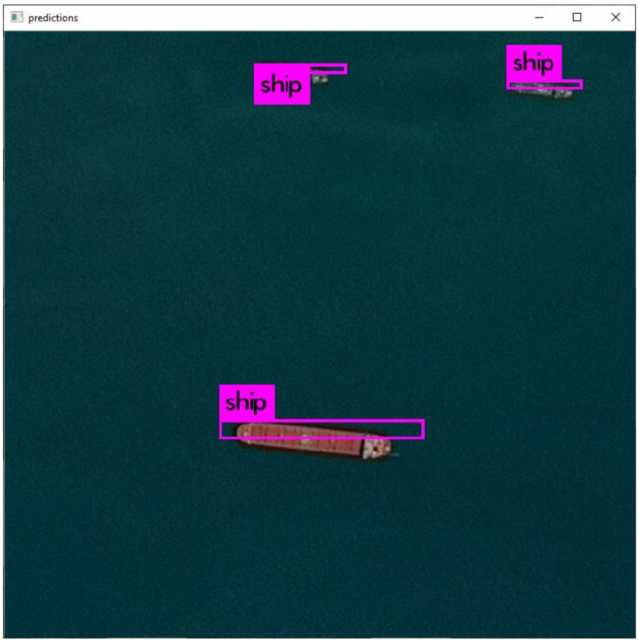


Moreover, a table comparing the metrics analyzed above is adduced. It is shown that the biggest difference occurs in recall, something that is clear from the sharp increase of false positive detections. The other two metrics though, have not experienced important deviations. This output occurs because the true positive predictions still exist and in some cases are more than those achieved with a higher confidence threshold.

Metrics	**t = 0.25**	**t = 0.10**
Precision	48%	35%
Recall	31%	41%
mAP	30.13%	34.20%

4.3 Ensemble Approach

The previous chart shows exactly our goal. We reduced the confidence threshold and we are now able to make more correct predictions, albeit with lower precision. However, we found out that these metrics can be a bit misleading as they are benchmarked based on the IoU meaning that if we predict a ship that the predicted bounding box falls outside the real bounding box, then this will be flagged as a false negative. However, in a region proposal context and the ensemble approach, we can pass the region through the classifier and find the ship regardless. This is clearly depicted in the below image the upper left ship's bounding box falls outside of the real bounding box and that leads to a false negative prediction thus reducing both the Recall and the Precision scores.



This threshold serves the purpose of collecting as many regions of interest as possible. Having these isolated image areas, we can apply our initial classifier directly on them, without having to scan the entire image, anymore. As a result, even though YOLO is not as accurate as we observed on the results above, it is used as a type of preprocessing that requires just 8.2 milliseconds and extracts regions of interest to pass through our previously trained CNN Tensorflow model. Then the CNN classifier receives these regions of interest, concatenates them into no more than 2-3 batches of 32 and infers in 2μs for each batch. Finally, have the boxes drawn and the whole image exported in under 1 second.

5 Conclusion

In this work, an end to end approach for ship detection in satellite images was developed. Also, it was shown how a dataset might need a lot of preprocessing augmentations until it can be used from a machine learning technique. The classifier and sliding window approach had the best recall and precision while greatly sacrificing running time while the All-CNN approach managed to quickly find ships, albeit with much lower accuracy. In order to get meaningful results, these two methods were combined as the YOLO detector was repurposed as a region proposal tool leaving the dedicated, high-accuracy classifier to operate on this subsection of the initial image.

It is also concluded that object detection in images pose a difficult task requiring a high amount of utilization resources, especially for small objects such is the case in ships seen in satellite images. Therefore, there is a need to propose methods that manage to reduce the solution to simpler steps if detection time is of high importance. Another important finding is the fact that when tuning the threshold that neural networks predict a given class, there is always a cost-benefit analysis on the recall and precision metrics. When thresholds become stricter than the recall metric is increasing at the cost of precision. On the contrary, when thresholds are being reduced, the recall is getting higher, but we get a lot of falser predictions in the sense of precision.

The final conclusion that can be extracted from this work is that a machine learning algorithm's success is limited by the quality of the data that it is fed. As seen in the Tensorflow classifier section, when a scene contained examples that did not have a similar case in the training dataset, the neural network could not correctly detect the ship. Moreover, having too few training samples impairs the ability of the algorithm to learn, as seen in the context of tidal barriers.

6 Future Work

The aforementioned study can benefit from implementing the other CNN based techniques and comparing them with YOLO. It is certain that YOLO is superior in terms of inference performance however it remains uncertain how it compares in term of mAP. However, it is safe to assume, from other datasets, that the different detector approaches behave similarly in these accuracy metrics.

Another task worth exploring is to implement an end to end YOLO-classifier implementation without relying on the underlying darknet framework. This will enable a better pipeline of data input/output and boost the proposed detector performance.

Finally, another important step for the robustness of the detector is how it performs when a variety of image alterations occur in the image. These alterations can fall under the category of noise or image transformations (rotation, expansion, etc.). Having a detector that is tested under these conditions would make the implementation more trustworthy for production level environments.

7 References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," *arXiv preprint arXiv:1603.04467,* 2015.

[2] C. Corbane, E. Pecoul, L. Demagistri and M. Petit, "Fully automated procedure for ship detection using optical satellite imagery," in *Proceedings of SPIE, the International Society for Optical Engineering*, 2008.

[3] C. Zhan, L. Zhang, Z. Zhong, S. Didi-Ooi, Y. Lin, Y. Zhang, S. Huang and C.-C. Wang, "Deep Learning Approach in Automatic Iceberg - Ship Detection with SAR Remote Sensing Data," in *International Geophysical Conference, Beijing, China, 24-27 April 2018*, 2018.

[4] D. P. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," in *ICLR 2015 : International Conference on Learning Representations 2015*, 2015.

[5] A. F. Agarap, "Deep Learning using Rectified Linear Units (ReLU).," *arXiv preprint arXiv:1803.08375,* 2018.

[6] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research,* vol. 15, no. 1, pp. 1929-1958, 2014.

[7] K. He, G. Gkioxari, P. Dollár and R. B. Girshick, "Mask R-CNN," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017.

[8] R. Girshick, "Fast R-CNN," *arXiv preprint arXiv:1504.08083,* 2015.

[9] S. Ren, K. He, R. B. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, no. 6, pp. 1137-1149, 2017.

[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu and A. C. Berg, "SSD: Single Shot MultiBox Detector," *european conference on computer vision,* pp. 21-37, 2016.

[11] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv preprint arXiv:1804.02767,* 2018.

[12] [Online]. Available: https://www.kaggle.com/rhammell/ships-in-satellite-imagery/home.

[13] K. Janocha and W. M. Czarnecki, "On Loss Functions for Deep Neural Networks in Classification," *Schedae Informaticae,* vol. 2016, p. 4959, 2017.

[14] [Online]. Available: https://www.kaggle.com/c/airbus-ship-detection/.

[15] [Online]. Available: https://github.com/AlexeyAB/darknet.