Informatik und Mathematik
Systems Engineering for Vision and Cognition
Goethe-Universität Frankfurt

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

MLPR Group Project

Ship Detection on satellite images using neural networks

Kaggle Challenge: Airbus Ship Detection Challenge

Submission Date: October 2, 2018

Team members

Hesam Ghavami
Iurii Mozzhorin
Hevin Özmen

Supervised by:

Prof. Dr. Visvanathan Ramesh
DBIS
Goethe University Frankfurt

Table of Contents

List of Figures

List of Tables

List of Abbreviations

FCN	Fully Convolutional Network
SGD	Stochastic Gradient Descent
ReLU	Rectified Linear Unit

1 Introduction

In light of the increasing size and demand in the ship transport sector the need for maritime monitoring services becomes pressing. This ensures the reduction of ship accidents as well as the mitigation of illegal activities such as piracy, drug trafficking, illegal fishing and movement of illegal cargo. There are a myriad of stakeholders interested in functioning maritime monitoring, such as government authorities, insurance companies as well as environmental protection organizations.

One organization specializing in solutions for advancing maritime monitoring, threat anticipation and triggering and improving overall efficiency is Airbus. One of the attempted advances initiated by Kaggle is the automatic ship detection from satellite images. The objective here is dedicated improvement in the dimensions of speed and accuracy, as these remain challenging in automatic object extraction from satellite imagery. This problem is published as a challenge on the machine learning competition community platform Kaggle Inc.[1] Airbus provided original satellite images and corresponding masks.

The authors of this report took part in this Kaggle competition as part of the machine learning project. The following report summarizes the results for ship detection from satellite images using different deep learning approaches.

Evaluation

This competition is evaluated on the F2 Score at different intersection over union (IoU) thresholds. The IoU of a proposed set of object pixels and a set of true object pixels is calculated as:

$$IoU\ (A, B) = \frac{A \cap B}{A \cup B}$$

The metric sweeps over a range of IoU thresholds, at each point calculating an F2 Score. The threshold values range from 0.5 to 0.95 with a step size of 0.05: (0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95). In other words, at a threshold of 0.5, a predicted object is considered a "hit" if its intersection over union with a ground truth object is greater than 0.5.

[1] https://www.kaggle.com/c/airbus-ship-detection#description, Accessed: September 03, 2018

At each threshold value t, the F2 Score value is calculated based on the number of true positives (TP), false negatives (FN), and false positives (FP) resulting from comparing the predicted object to all ground truth objects. The following equation is equivalent to F2 Score when β is set to 2:

$$F_\beta(t) = \frac{(1+\beta^2)\cdot TP(t)}{(1+\beta^2)\cdot TP(t) + \beta^2 \cdot FN(t) + FP(t)}$$

A true positive is counted when a single predicted object matches a ground truth object with an IoU above the threshold. A false positive indicates a predicted object had no associated ground truth object. A false negative indicates a ground truth object had no associated predicted object. The average F2 Score of a single image is then calculated as the mean of the above F2 Score values at each IoU threshold:

$$\frac{1}{|thresholds|}\sum_t F_2(t)$$

Lastly, the score returned by the competition metric is the mean taken over the individual average F2 Scores of each image in the test dataset. https://www.kaggle.com/c/airbus-ship-detection#evaluation

Prizes

- 1st place - $25,000
- 2nd place - $15,000
- 3rd place - $5,000
- Algorithm Speed Prize (Post competition prize) - $15,000

The goal of the Algorithm Speed Prize is to encourage participants to develop the fastest running algorithm that will still produce accurate results. The speed prize will be open to the top performing teams on the leaderboard.

Timeline

- July 30, 2018 - Start Date.
- September 27, 2018 - Entry deadline.
- September 27, 2018 - Team Merger deadline.
- October 4, 2018 - Final submission deadline.
- October 18, 2018 - Final submission deadline for the Algorithm Speed Prize.
- November 22, 2018 - Announcement of the winner of the Algorithm Speed Prize.

Because of a data leak, the deadlines were extended. Details on the data leak are provided in Section 3 (Data Understanding).

2 Methodology

Since the idea of the challenge was not only to achieve a good accuracy in ship detection and localization but also develop a quicker high-performance algorithm, we have suggested using some extra technic for this purpose. We wanted to compare two models, which are widely used for image segmentation and object localization: Mask R-CNN and U-net. U-Net was developed for the biomedical data but recently has shown good results for satellite images also. Mask R-CNN is a successor of Fast R-CNN and Faster R-CNN and now is officially a part of a Facebook's project Detectron. Both of these models have a deep architecture and many specific heuristics to improve their precision. Both models are characterized by a big number of parameters, what makes them relatively slow in training and predictions. Thus, we have tried to combine them with more simple and fast (pre)classification method. In the first stage this method should scan through the images and reject the empty ones, so on the second stage, our large models will process only the selected images, which contain ships with some high probability. Consequently, we aimed to compare the accuracy and performance of Mask R-CNN and U-net on the whole test data and only on the selected during the preclassification part. Due to some technical difficulties, we have not combined preclassification and localization blocks in one model, so the estimations of the performance improvements are based on the ratio of images, selected during the preclassification, with the whole amount of the test data. That should be considered as a low boundary of the performance improving because does not include possible parallel processing of the data through these two stages.

3 Data Understanding

The original train dataset contained 104700 satellite images with 131030 ships on them. Test set contained 88486 images. 15 images were corrupted (1 from the train and the rest from the test). All images have resolution 768x768 px and were cut from some bigger images. No location or time data were provided in the challenge. Kaggle had reassured that there is no overlapping in the images.

Besides ships and water, the images also contain clouds, haze, wakes behind the ship (that will not count as part of the ship), coastal areas, docks, marinas, reflections, waves and other floating objects such as buoys, barges, wind turbines, etc. There is also some image cutting artifacts.

Most of the images are empty, only ~39% of them contains ships. You can see on the histogram that images with only one ship are dominating.

After a few weeks of running a severe data leak in the challenge was found: test images occurred to be just shifted train images, there were also overlaps inside train set. Then the challenge hosts provided masks also for the test set, so it can be also used for training. The new test set should be provided on the first week of October. Until that we have focused on training and validating the old data.[2]

The new test set was published on 8th October. It contains only 15606 images.

[2] https://www.kaggle.com/c/airbus-ship-detection/discussion/64388, Accessed: September 05, 2018

4 Preclassification

We aimed to compare several different methods for preclassification but, because of some technical difficulties and a shortage of time, we have focused only on simple models of convolutionally neural networks.

4.1 Data Preparation

There was no complete data loader for these challenge based on Pytorch, which includes all the required functions, so we developed our own using examples from the practice week and some Kaggle kernels. During the preprocessing, the images were resized and augmented. The corrupted images and images with the size smaller than 40 KB (~200 images) were dropped. 10% of the train data set was used only for validating during training.

Resize factor

In order to accelerate the preclassification we have used instead of original 768x768 px images the resized versions of them:

- factor 2 (384x384 px)
- factor 4 (192x192 px)
- factor 8 (96x96 px)

Since some ships take only a few percents of the original images square, with the resize factor 8 they will take only several pixels and will be lost by classification. So after a short check, we rejected this factor for the detailed research. With the resize factor 2 both models have converges to either always 1 or always 0.

Data Balancing

One of the technics, which we have tried, is the balancing the ratio of negative examples using only 30% of empty images from train data and all the images with ships on them. In this case, the amount of positive and negative examples are approximately equal. That heuristic can be considered as not clean in data science because the structure of the train data supposed to be similar to test data, but in combination with a specific threshold on a softmax function, it gave us a suitable result for the resize factor 4. Unfortunately, for the resize factor 2 that did not help with models convergence to extreme states.

4.2 Data Modelling

We tried different architectures and settled on two models with two convolution layers with kernels 5x5 and one linear layer:

- 8 and 16 features (CNN_8)
- 32 and 64 features (CNN_32)

Each convolutional layer was followed by ReLU function and max-pooling with kernel 2x2. We have used cross-entropy as a loss function and stochastic gradient descent for optimization. After few tries, we chose the learning rate 0.001, the momentum 0.9 and the weight decay 0.0005. We have trained the models until the loss stops descending, that was after about 50 epochs.

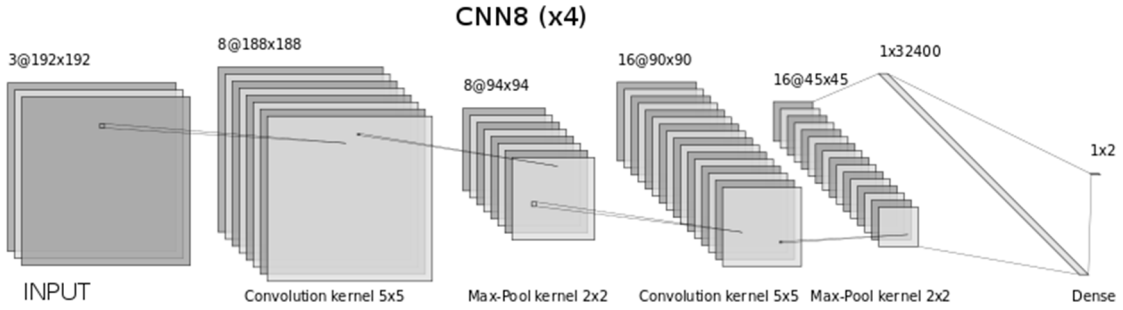


Figure 1: Network architecture of CNN8

We have also compared the usage of these models with the different image resize factors and balance of positive and negative examples for the training. And in the end, we have compared several of these combinations setting different thresholds on softmax functions. For this work, we have used Pytorch.

Training

Here we show some curves of the loss function and accuracy. We have used a simple binary accuracy: 1 if the prediction is right and 0 else. For our task it is not the optimal metric, so we have mostly focused on the confusion matrices. You can see how their values change during the training. Here we have only two types of errors:

- false positive, when the model predicts ships on empty images,
- false negative, when we lose some images with ships in the next stage.

The first brings down the efficiency of the algorithm and the second - its precision. So in any case, our task is to find a trade-off between these two characteristics. We assume that the minimization of the false negative error has here a higher priority. Previously we have predicted a class of an image just taking one of them with a maximal weight. For more precise models tuning we have passed the weights through the softmax function, which returns us values in [0,1], and compared results for different thresholds. The previous predictions will be similar to a threshold 0.5. We have decided that the optimal ratio between two errors gives us CNN_8 model with resize factor 2 trained on balanced data with the threshold 0.4.

Predicting

After choosing the best model and resize factor we have run the preclassification on the old test data (~88k images) for two thresholds: 0.4 and 0.5. As we have previously described a data leak occurred in the challenge, so the hosts have provided masks also for this test set. According to these data we have evaluated our predictions one more time more accurate. Using 0.4 threshold we lose only ~5% of the images with ships on them but rejecting ~35% of images, thus, the performance of the next stage can be improved minimum at 1.53 times comparing to processing the whole dataset.

We did the same also for the new test set (~15k images), but in this case, the quality of classification can be observed only indirectly through the score after the second phase.

4.3 Technical Specifications and Runtime

The model has run on an Amazon Web Service (AWS) p2.xlarge instance with the following specifications using GPU (CUDA) acceleration:

- 4 Core CPU (Intel Xeon E5-2686 v4 Broadwell)
- 61 GB RAM
- 1 GPU (NVIDIA K80 with 2496 cores and 12 GB memory)

One run on the old test data set (88486 images) takes on average 9:25 (min:sec) and on the new one (15606 images) - only 85 seconds, which is much faster than both models need on the second stage.

The code specifications for data preparation, modelling and visualization are provided in the Appendix (see 9.1 Code specifications for preclassification).

4.4 Data Visualization and Evaluation

Loss

The loss was recorded during training and validation for both models respectively.

- 8 & 16 features (CNN_8)
- 32 & 64 features (CNN_32)

The loss function for CNN_8 for balanced (cnn_8_x4_b) and unbalanced (cnn_8_x4_nb) datasets is depicted in Figure 2.

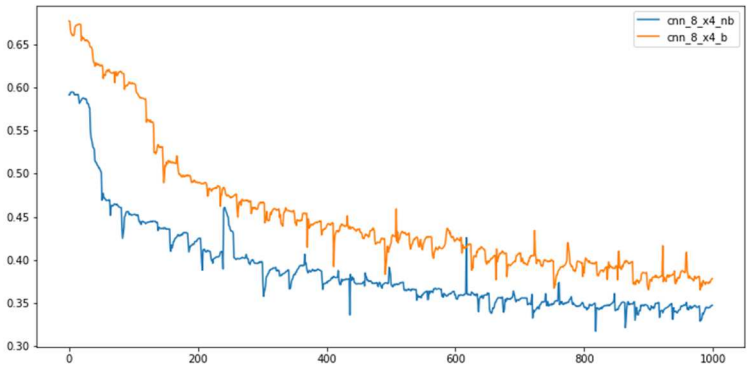


Figure 2: Loss (Cross-Entropy) over steps for balanced and unbalanced set

The loss function for CNN_32 for unbalanced dataset is displayed in Figure 3

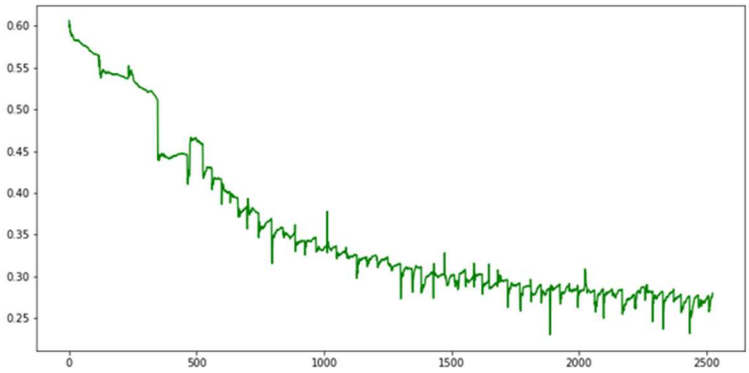


Figure 3: CNN_32 Unbalanced | Loss (Cross-Entropy) over steps

Accuracy

The accuracy for CNN_8 for each epoch is presented in Figure 4.

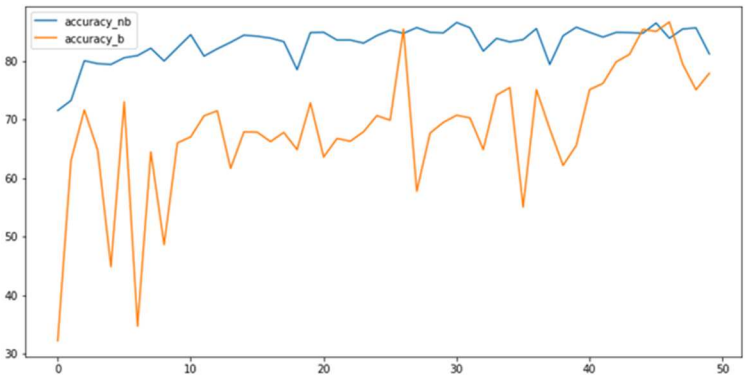


Figure 4: CNN_8 Accuracy per epoch for balanced and unbalanced set

Errors

Errors in predictions are shown in the following. Figure 5 plots these errors (false positives and false negatives). Blue and orange represent the functions for false positives for unbalanced and balanced respectively, and green and red depict false negatives for unbalanced and balanced respectively.

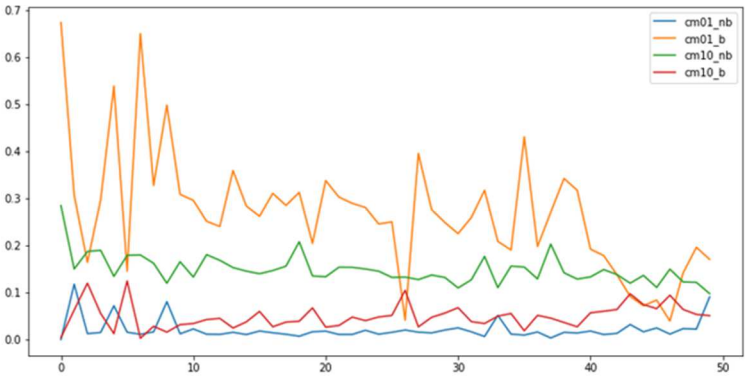


Figure 5: CNN_8 Errors per epoch for balanced and unbalanced set

The Confusion matrix displayed in gives further insight into the depicted errors.

Table 1: Confusion matrix for Preclassification

Predicted		CNN8_nb		CNN8_b		CNN32_nb	
		0	1	0	1	0	1
True	0	0.62	0.09	0.54	0.17	0.69	0.02
	1	0.10	0.19	0.05	0.23	0.11	0.17

Accordingly, the softmax thresholds for the erros are presented in Figure 6. The blue and orange line represent the errors for the unbalanced set, for false positives and false negatives respectively. The green and red line plot the balanced error function for false positives and false negatives.

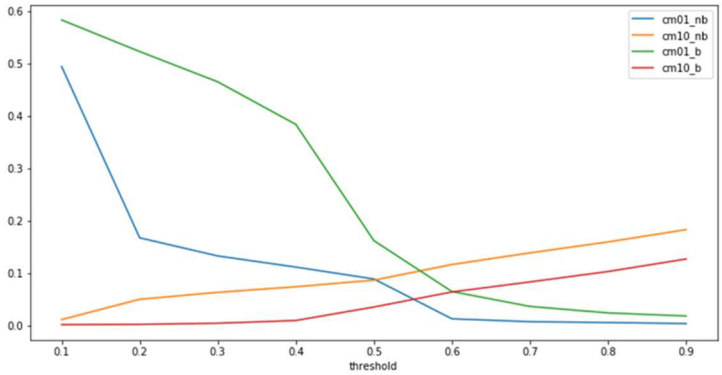


Figure 6: CNN_8 Errors per threshold for balanced and unbalanced set

5 U-Net

5.1 Data Preparation

The definition of hyperparameters is performed prior to the data preparation. The definition is highly dependent on technical characteristics of the used architecture. The used modules are general statistical libraries and modules such as numpy, pandas and os. As the model is performed on the Keras Python Deep Learning API with TensorFlow backend.

The data preparation phase included the following steps: run length encoding and decoding, transformation of masks to images, preparation of the train and validation set, data balancing and data augmentation.

5.2 Data Modelling

The selected model architecture is a U-Net, a convolutional neural network (CNN) with skip connections located prior to the bottleneck layer. It consists of Convolution Operation, Max Pooling, ReLU Activation, Downsampling/Concatenation and Upsampling/Expanding Layers. (Ronneberger et al. 2015)

The U-Net architecture is built upon the Fully Convolutional Network (FCN) and modified in a way that it yields better segmentation in medical imaging. U-net is symmetric and the skip connections between the downsampling path and the upsampling path apply a concatenation operator instead of a sum (compared to FCN). These skip connections intend to provide local information to the global information while upsampling. Because of its symmetry, the network has a large number of feature maps in the upsampling path, which allows to transfer information.

The architecture can be divided into three parts, the downsampling path, the upsampling path and the bottleneck. The former consists of four blocks, each represented as two 3x3 convolution layers and a corresponding activation function and 2x2 Max Pooling. The images and corresponding masks are ingested into the model, starting with 64 feature maps for the first block. The number of feature maps is then doubled at each pooling. The downsampling path is used to encode the context of the input as the basis for segmentation. This context is passed to the upsampling path using skip connections between down- and upsampling blocks.

The upsampling path consists of two 3x3 convolution layers with corresponding activation functions, concatenation with the feature map from downsampling, and a deconvolution

layer of stride 2. This path enables precise localization going up the architecture and combined with the contextual information from concatenation, transferring the information's state before being passed to the bottleneck.. Finally, the bottleneck is located in between both sampling paths and is built on two convolution layers. The general idea behind this symmetric architecture is to concatenate higher resolution features from contracting and combine them with the sampled features for better pixel-based localization. The basic architecture of U-Nets is presented in the Appendix (see Section 9.2).

The U-Net model was implemented using the Keras Sequential model (Doumentation: Keras). Keras is a deep learning library based on Python programming language and can run on TensorFlow and Theano, among other backends. The implementation described here used TensorFlow backend. The model is implemented by calling the **`Sequential()`** model, and then simply adding the U-Net layers using the **`.add ()`** method. After modelling is finished, the learning process can be implemented using the **`.compile()`** function. The training and validation is based on the following parameters:

- Loss function: Binary Cross-Entropy Loss
- Optimizer: Stochastic Gradient Descent
- Accuracy Meter: Keras Binary Accuracy
- Additional additional metric: Dice Coefficient

The model was trained using a balanced train sample. The predictions were then made in two phases:

I. Prediction for the entire test data set.

II. Prediction only for the images, that were identified as images containing ships in the Preclassification Model (Section 4)**.**

5.3 Technical specifications

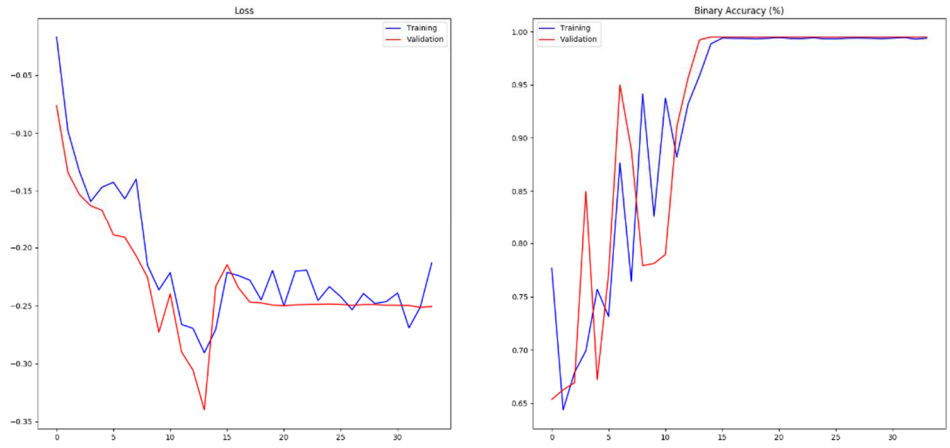
[TBD]

5.4 Data Visualization and Evaluation

Extract of validation (see Appendix)

Description

Evaluation of loss history and binary accuracy per epoch:



Extract of prediction (see Appendix)

Submission of predictions for entire test data

Various submissions were made to Kaggle. The scores were as follows

1. Submission October 14, 2016: early stopping after 34 epochs, 50 maximum epochs were defined, optimizer: Adam, loss function: IoU, steps per epoch: 50 steps

   Kaggle accuracy score: **0.546**

2. Submission October 15, 2016: early stopping after XX epochs, 50 maximum epochs were defined, optimizer: SGD, loss function: Binary Crossentropy, steps per epoch: 50 steps

Submission of predictions for preclassified test data

1. Submission October 15, 2016:
2. Submission October 15, 2016:

6 Mask R-CNN

6.1 Data Preparation

6.2 Data Modelling

6.3 Data Visualization and Evaluation

6.4 Evaluation and Summary of the findings

In the end, we can compare performance (runtime) and accuracy (score from the challenge) for all three solutions

7 Evaluation and Summary

8 References

Doumentation: Keras: Keras Documentation. Keras: The Python Deep Learning library | Model class API . Online verfügbar unter https://keras.io/models/model/, zuletzt geprüft am September, 2018.

Ronneberger, Olaf; Fischer, Philipp; Brox, Thomas (2015): U-Net. Convolutional Networks for Biomedical Image Segmentation.

9 Appendix

9.1 Code specifications for preclassification

Python programms:

- airbus_dataloader.py - data loading and preprocessing
- airbus_models.py - used models (CNN_8 & CNN_32)
- airbus_train_val_functions.py - set of functions for training/validating and accuracy calculating
- airbus_train.py - train programm
- airbus_compare_thresholds.py - compare different thresholds on softmax function
- airbus_preclassification.py - generating the predictions

Jupyter notebooks:

- data_exploration.ipynb - dataset visualisation
- preclassifier_predictions.ipynb - generating the predictions
- preclassifier_predictions_analysis.ipynb - visualization of the results of predicting
- cascade_classifier_datapreparation.ipynb - datapreparation for a cascade classifier

Other files:

- ccc_*.py - versions for the CCC's cluster (different folders)
- ccc_negative_exaples.txt, ccc_positive_exaples.txt - dataset for the training a cascade classifier
- *.model - saved weights for models
- filtered_ships*.txt - results of the preclassification for the next stage

9.2 U-Net Architecture

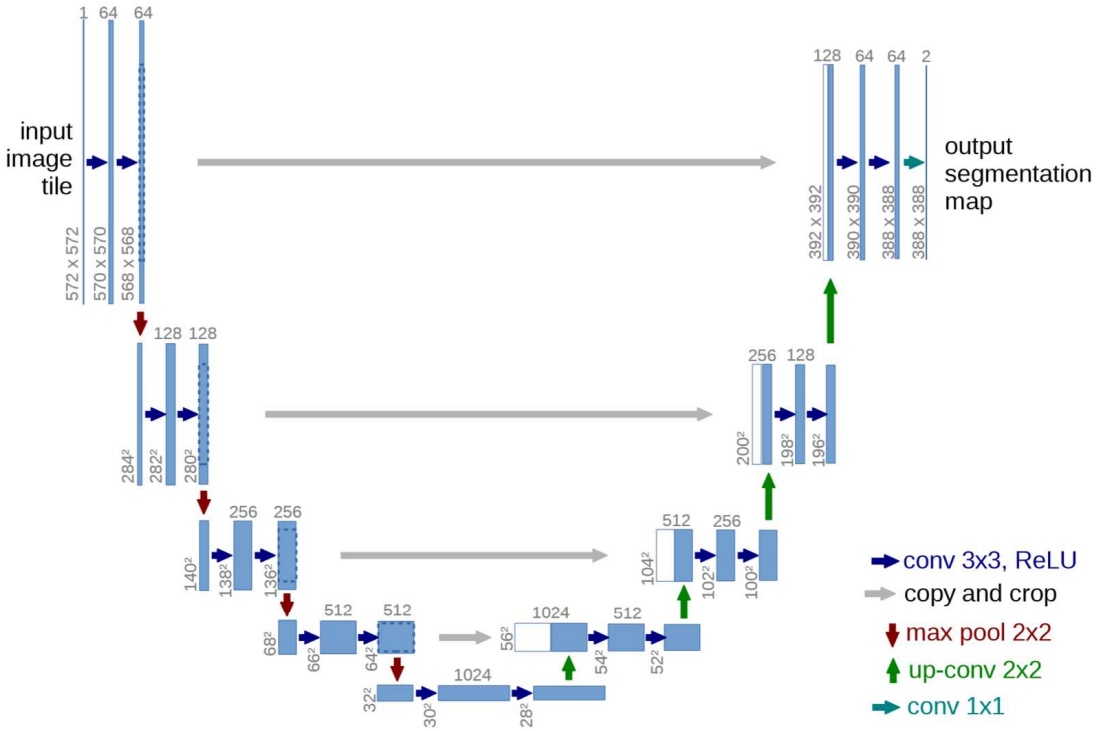


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Source: (Ronneberger et al. 2015)