University of Warsaw
Faculty of Physics

Recognition of satellite images of clouds using deep learning

Wiktor Kondrusiewicz, Jan Franciszek Klamka, Marta Włodarczyk,
Maciej Kierkla, Tomasz Bednarek

under the supervision of dr hab. Jarosław Żygierewicz

June 24, 2020

Contents

1 Introduction

The aim of this project was to participate in a Kaggle competition called 'Understanding Clouds from Satellite Images' [1] hosted by Max Planck Institute for Meteorology. Its purpose was to classify and segment masks of clouds on satellite images. We did it as a university team project to broaden our knowledge and acquire experience in applying deep learning to real-world problems. Thus, our primary aim was to learn new machine learning techniques, whereas pursuing the best metrics was considered secondary. The competition is motivated by the climate change that should be at the top of our minds and on the forefront of important political decision-making for many years. Scientists from Max Planck Institute for Meteorology hope that this competition's dataset will help demystify an important climatic variable.

Shallow clouds play a huge role in determining the Earth's climate. They're also difficult to understand and to represent in climate models. By classifying different types of cloud organization, researchers at Max Planck hope to improve our physical understanding of these clouds, which in turn will help us build better climate models.

There are many ways in which clouds can organize, but the boundaries between different forms of organization are murky. This makes it challenging to build traditional rule-based algorithms to separate cloud features. The human eye, however, is really good at detecting features—such as clouds that resemble flowers.

2 Used approaches

2.1 Data preparation

Data provided by the authors of the competition [1] consisted of 5546 satellite images of clouds that were used for training and validation of the models. There were also 3698 images that were reserved for the test dataset, but **we did not take them into account as they were used to evaluate the performance of the model on Kaggle competition and we focused on learning as it was a university project**. The images were in 1400x2100 resolution and could contain up to 4 different cloud types: Fish, Gravel, Sugar and Flower.

The task that the models were supposed to tackle is called **semantic segmentation**, which accounts for predicting pixel mask for each class type (here for each of 4 clouds classes) - also the model needs to correctly predict the mask's class. Thus it can be seen as two-fold task; generating cloud's mask and correctly classifying it to one of 4 aforementioned classes.

The masks for clouds were supplied in a spreadsheet like file (more precisely a .csv file). But because of the size of the images masks were represented using running-length encoding (RLE) on the pixel values. It requires that one submits pairs of values that contain a start position and a run length. For example, '1 3 10 5' implies pixels 1,2,3,10,11,12,13,14 are to be included in the mask. The pixels are numbered from top to bottom, then left to right: 1 is pixel (1,1), 2 is pixel (2,1), etc. Owing to the RLE, the size of file with encoded masks is $\approx$ 200 MB instead of tens of GB.

We established helper functions for decoding RLE to masks and then passing pairs of (image, corresponding masks) to our models. So to reproduce our results user only needs to download the dataset from the competition website [1] and clone our GitHub repository [2].

2.2 Evaluated neural networks

The whole code base was written using Python programming language with the support of PyTorch [3] library (version 1.4.0), which is one of the most popular frameworks used for deep learning. It offers already pre-trained networks, such as Mask R-CNN (discussed in Sec.2.2.2) and many other convenient tools that facilitate process of developing deep learning solutions. Our project also uses other external libraries that are listed in `requirements.txt` file in the repository [2].

2.2.1 U-Net

The U-Net [4] architecture stems from the so-called 'fully convolutional network' (FCN for short) which uses only convolution operations (in contrast to many neural networks that incorporate some dense layers). The fundamental idea is to augment a normal encoder network with a decoder. Namely, the encoder that captures patterns in the data and extract features into the output feature map is followed by the decoder that uses transposed convolutions (also called up-convolutions or upsampling layers) to process the feature map into the image that has identical size as the input to the encoder.

U-Net uses a alrge number of channels in the upsampling part that allows the network to properly pass the information to higher resolution layers. Additionally, the upsampling part is quite symmetric to the contracting one and it resembles u-shaped architecture, hence the name 'U-Net'.

It was first used to segment biomedical images obtained from various scanners (e.g MRI) and it achieved state-of-the-art performance. Thus, we decided to give it a try in terms of segmenting cloud images.

U-Net has n output channels for n classes, where each channel contains a binary mask for a given class - in our case $n = 4$. The network architecture is presented in Fig. 1. The implementation was derived from [5], [6]. Yet, there were few differences, namely:

- used image input size of 528×352 (we obtained this from the original image using image resizing function from OpenCV library [7]) instead of 256×256 depicted in Fig. 1. It was due to the fact that implementation from [5] did not use adaptive padding, so the input size had to be divisible by 16,
- the last layer, showed in upper right corner of Fig. 1 in our case was of size `256x256x4` (keeping the input image size as in the picture), where 4 denotes the number of clouds classes.

After implementing the changes above the network had $7.8 * 10^6$ (or more precisely 7847236) trainable parameters (weights).

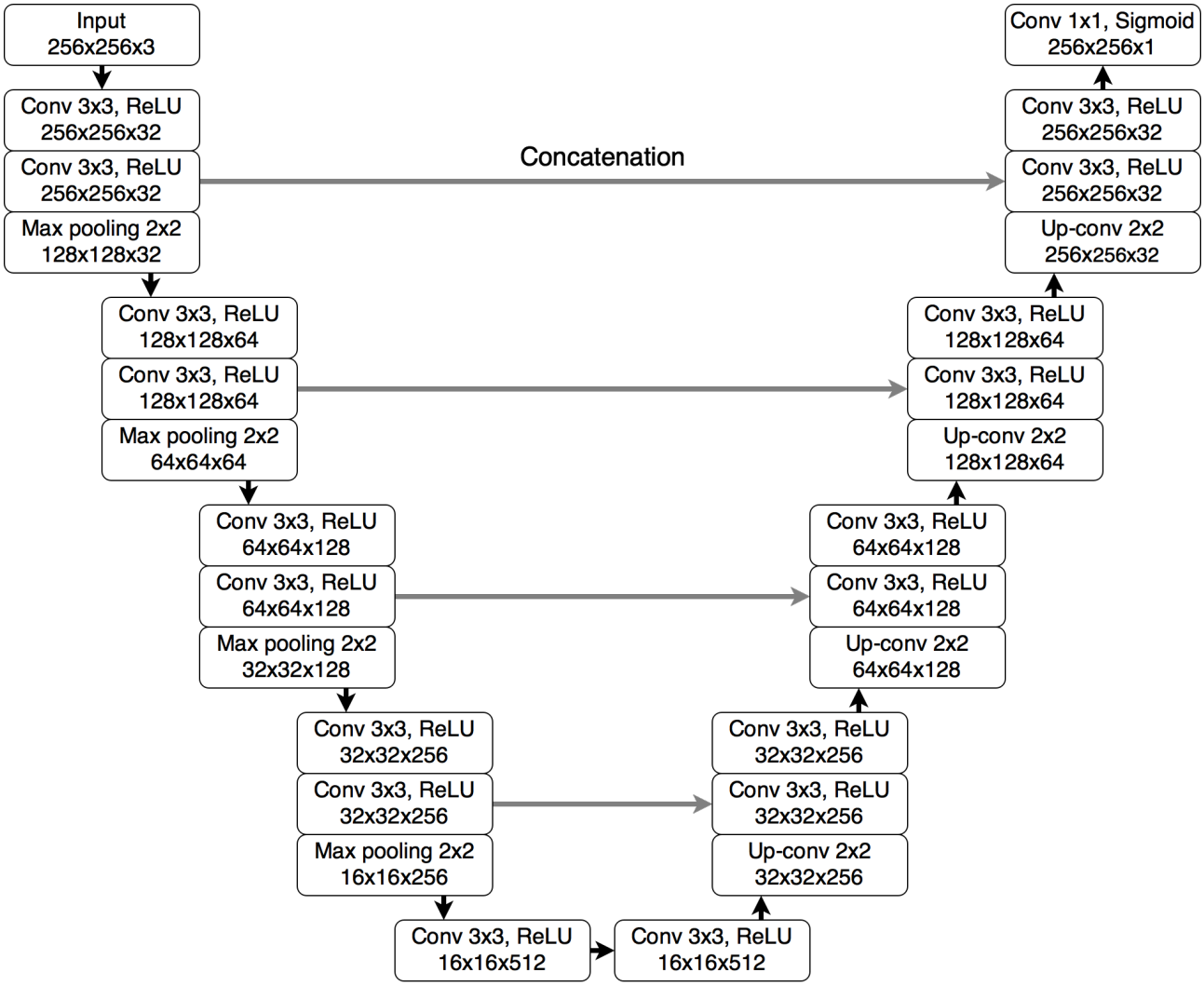

Figure 1: Scheme of U-Net's architecture used in this project [5]. Size of input image vary from the one showed in this figure (we used 528×352). Also we had to modify the last layer (upper right corner) to be of size `256x256x4` (keeping the input image size as in the picture), where 4 denotes the number of clouds classes.

The network was initialized with already pretrained weights on the task of "FLAIR (Fluid-attenuated inversion recovery) abnormality segmentation in brain MRI" as discussed in [5]. Also, as mentioned above, the last layer of the network was replaced with convolutional layer with 4 filters instead of 1 (leaving the rest unchanged, so the last activation was still Sigmoid function as presented in Fig. 1). Then, the whole network was trained on the dataset from Understanding Clouds competition.

2.2.2 Mask R-CNN

Mask R-CNN [8] is a conceptually simple, flexible, and general framework for object instance segmentation. It is an extension of Faster R-CNN [9] by adding a branch for predicting an object's mask in parallel with the existing branch for bounding box recognition. It is clearly visible in Figures. 2 and 3.

Figure 2 presents scheme of Faster R-CNN architecture that is excellent for object detection (detecting bounding boxes of objects in the image). Succinctly, Faster R-CNN works as follows:

1. Firstly it uses deep convolutional network (usually some variant of ResNet [10]) to obtain feature maps from the image,
2. Then, they are fed into Region Proposal Network (RPN) that gives bounding boxes proposals - its main objective is to predict if any object is present in the bounding box
3. Afterwards Region of Interest (RoI) pooling is applied (basically it works as adaptive max-pooling) to ensure that all regions have the same shape. A region is considered positive if its IoU (Intersection over Union, basically ratio of area of intersection of region and ground truth bounding box to its union area) is greater than 0.5,
4. At last, candidates are fed into dense network that classifies objects and outputs corresponding bounding boxes.

As mentioned, Mask R-CNN adds additional branch that uses FCN (fully convolutional network) on each of selected regions of interest (the ones that were positive) to acquire a segmentation mask. It is best illustrated in Fig. 3.

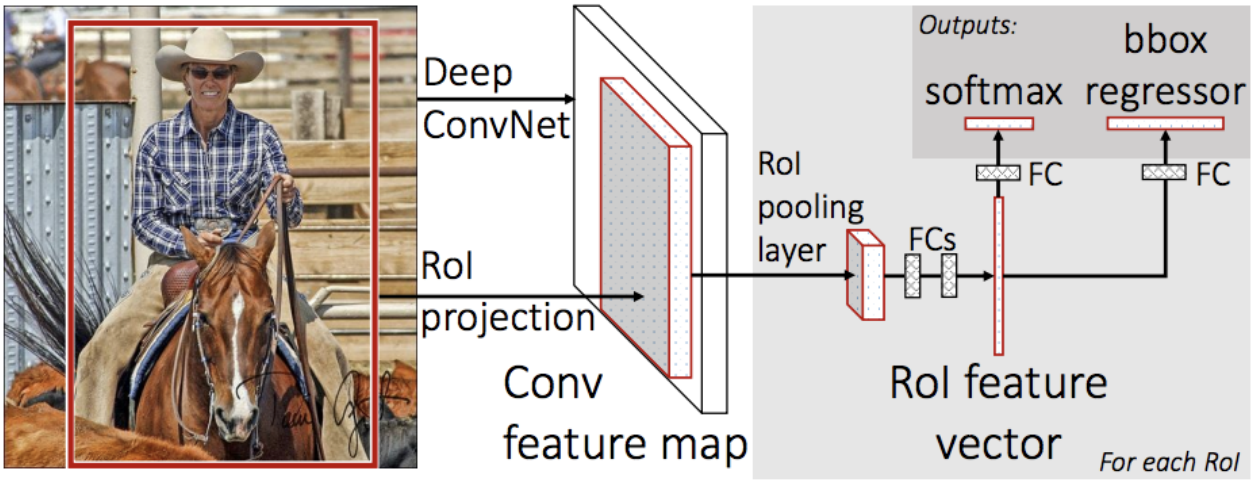

Figure 2: Scheme of Faster R-CNN architecture that is used as a building block for Mask R-CNN. Source link

It was chosen as one of the networks to try as it achieves state-of-the-art performance in instance segmentation tasks. Although the task of this project was focused at semantic segmentation we decided utilize Mask R-CNN because of two reasons, namely:

1. instance segmentation is a generalization of semantic segmentation and basing on the class of the predicted mask we can merge all masks corresponding to one class into single mask,
2. we wanted to learn how to use and harness such an advanced and powerful model in a real world machine learning project.

Also, getting acquainted with the Mask R-CNN architecture helps to use similar architectures like its version for estimating key-points in the image (e.g useful for pose estimation).

We used Mask R-CNN model that was pretrained on the COCO dataset (a large-scale object detection, segmentation, and captioning dataset with over 330 thousand labeled images, source link). It is available in PyTorch library, and for our task the version of Mask R-CNN had $4.3 * 10^7$ (more precisely 43716141) trainable parameters (weights). The feature maps extractor used as the backbone in this version of Mask R-CNN was ResNet 50. The output of mask branch of Mask R-CNN had 5 channels as there were 4 classes of clouds to be predicted and the models itself requires one additional class for background (when there is no object present).

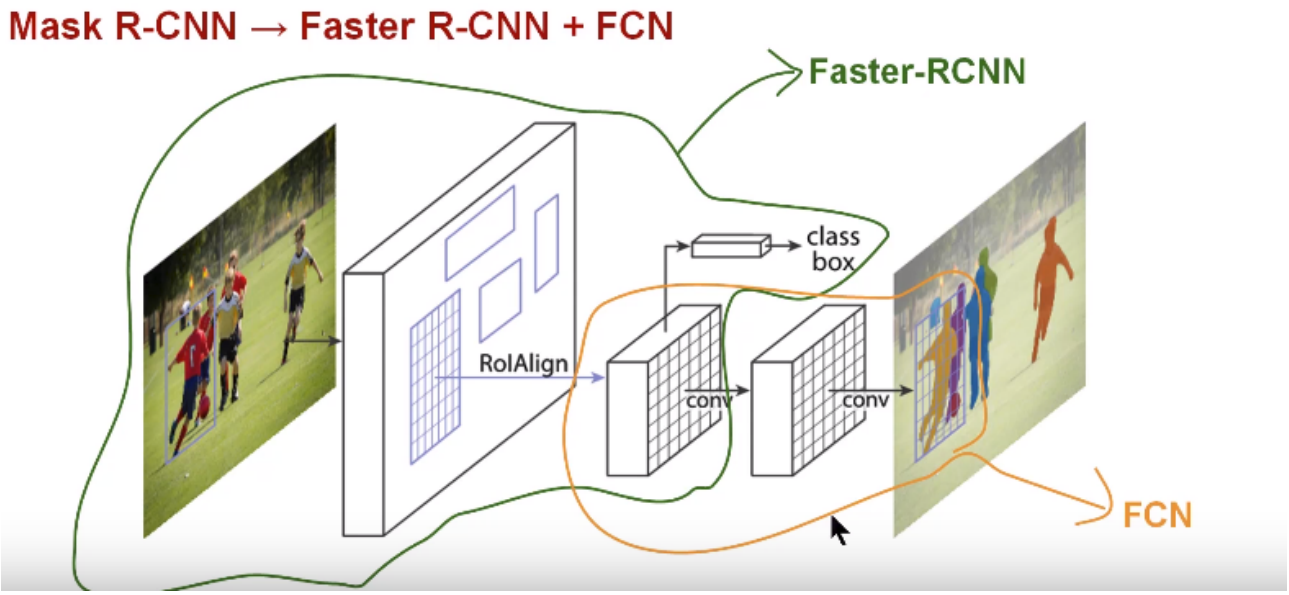

Figure 3: Scheme of Mask R-CNN architecture built on top of Faster R-CNN architecture. Source link

We used image input size of 525×350, which is 4 times smaller than the original size. We obtained this from the original image using image scaling function from OpenCV library [7].

2.3 Evaluation metrics

Authors of the competition stated that Dice Coefficient $D(X, Y)$ (also known as Dice Score) [11] should be used as a primary evaluation metric. It can be used to compare the pixel-wise agreement between a predicted segmentation and its corresponding ground truth. It is defined as follows:

$$D(X, Y) = \frac{2 * |X \cap Y|}{|X| + |Y|}, \tag{1}$$

where X denotes predicted set of pixels, Y corresponds to the ground truth data and $|\cdot|$ describes cardinality (number of elements) of a set. In case of many masks for one image it is calculated separately for each (image, mask) pair and then averaged. Also, our implementation of dice coefficient **does NOT consider empty masks as the authors of the competition wanted because it significantly increased the metric's value, yet it did not correspond to real-world performance**; it could be done in such matter because we did not publish results in the Kaggle competition.

3 Results

A typical approach to analyze behavior of a neural network is to plot its loss function's value as a function of training epochs (1 epoch correspond to iterating over an entire dataset of images). It is also beneficial to simultaneously check network's performance on a small subset of a training set that the network has not seen before - it is called a validation set.

In our case, to create a validation set we randomly chose 5% of a training dataset with the help of `train_test_split()` function from `scikit-learn` library [12]. As mentioned in Sec.2.1 we did not use the test set provided by the authors of the competition so all of the generalizing capabilities of the models were reported on the validation set obtained using the aforementioned `scikit-learn` function. Also, we did not take k-fold cross validation (CV) into account as the dataset size was too big to train 4-5 models (this would be the amount of folds that would makes sense for using CV). **Yet, we realize that we could have saved some small (even smaller than for validation set) portion of the validation data to measure networks performance even better**. After every epoch, a network was supplied with validation dataset to measure its loss.

3.1 U-Net

U-Net was trained for 350 epochs on Nvidia GTX 1070 graphics card. The training was composed of three consecutive runs with the following parameters (after each run network's weights were save on a disk and then

restored for the next run):

- 100 epochs with learning rate of $1 * 10^{-3}$, weight decay (L2 regularization) of 0.005 and exponential learning rate decay coefficient of 0.98,
- 100 epochs with learning rate of $1 * 10^{-4}$ learning rate decay coefficient of 0.96 and the rest of parameters as mentioned above,
- 150 epochs with learning rate of $5 * 10^{-6}$, weight decay of 0.98 and rest of parameters unchanged.

Also, throughout the entire training we used batch sizes of 8 images for training and 2 for validation.

Performance of the model can be observed in Fig. 4 that depict the total loss for training and validation sets. One can observe that at the training loss was decreasing for first epoch 100 epochs, yet the validation curve is purely oscillating between quite large values thus we can conclude that the network faced heavy overfitting. So, network's generalizing properties are not satisfying.

Yet, figures 5, 6 present the examples where the dice score was **the highest** for training (≈ 0.798) and validation (≈ 0.754) sets respectively and they do not look terrible. However, those results are cherry picked and the overall performance of the mdoel is reflected in the general scores, namely: dice coefficient of 0.244 for whole training set and 0.248 for validation respectively. Such discrepancy between the results for the best examples and average for the entire dataset arises from the fact that the network performed poorly for some classes (namely Flower). Comments of the possible improvement can be found in Sec.4.

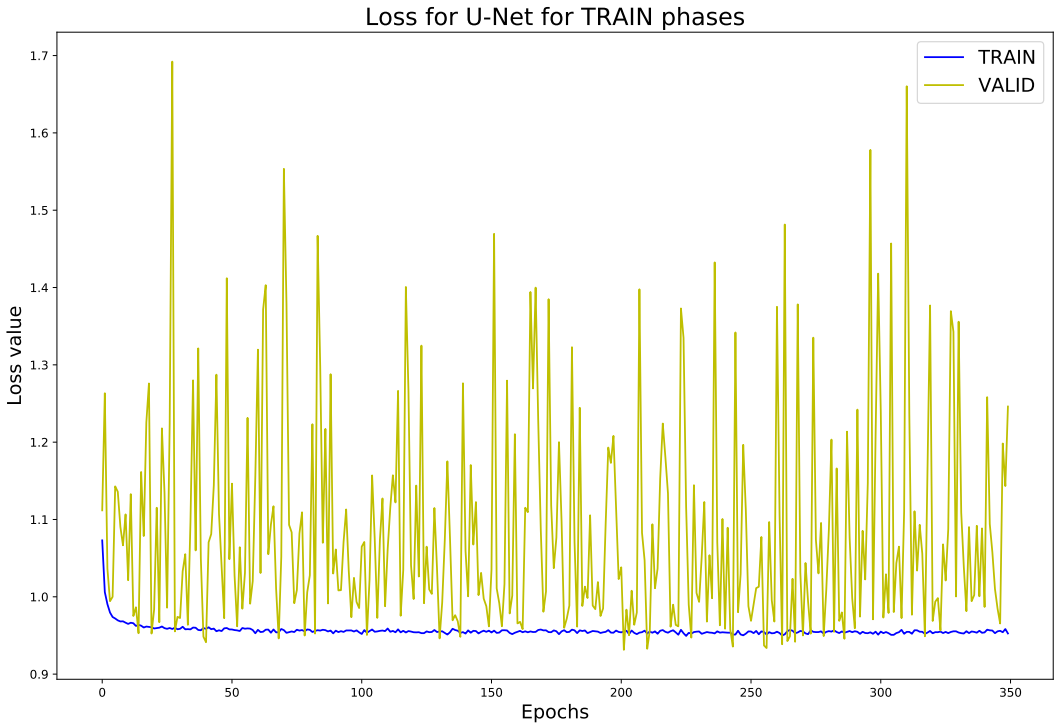

Figure 4: Loss of U-Net for training (blue) and validation (yellow) phases for 350 epochs.

3.2 Mask R-CNN

Mask R-CNN was trained for 350 epochs on Nvidia GTX 1070 graphics card. The training was composed of three consecutive runs with the following parameters (after each run network's weights were save on a disk and then restored for the next run):

- 100 epochs with learning rate of $1 * 10^{-5}$, weight decay (L2 regularization) of 0.005 and exponential learning rate decay coefficient of 0.98,

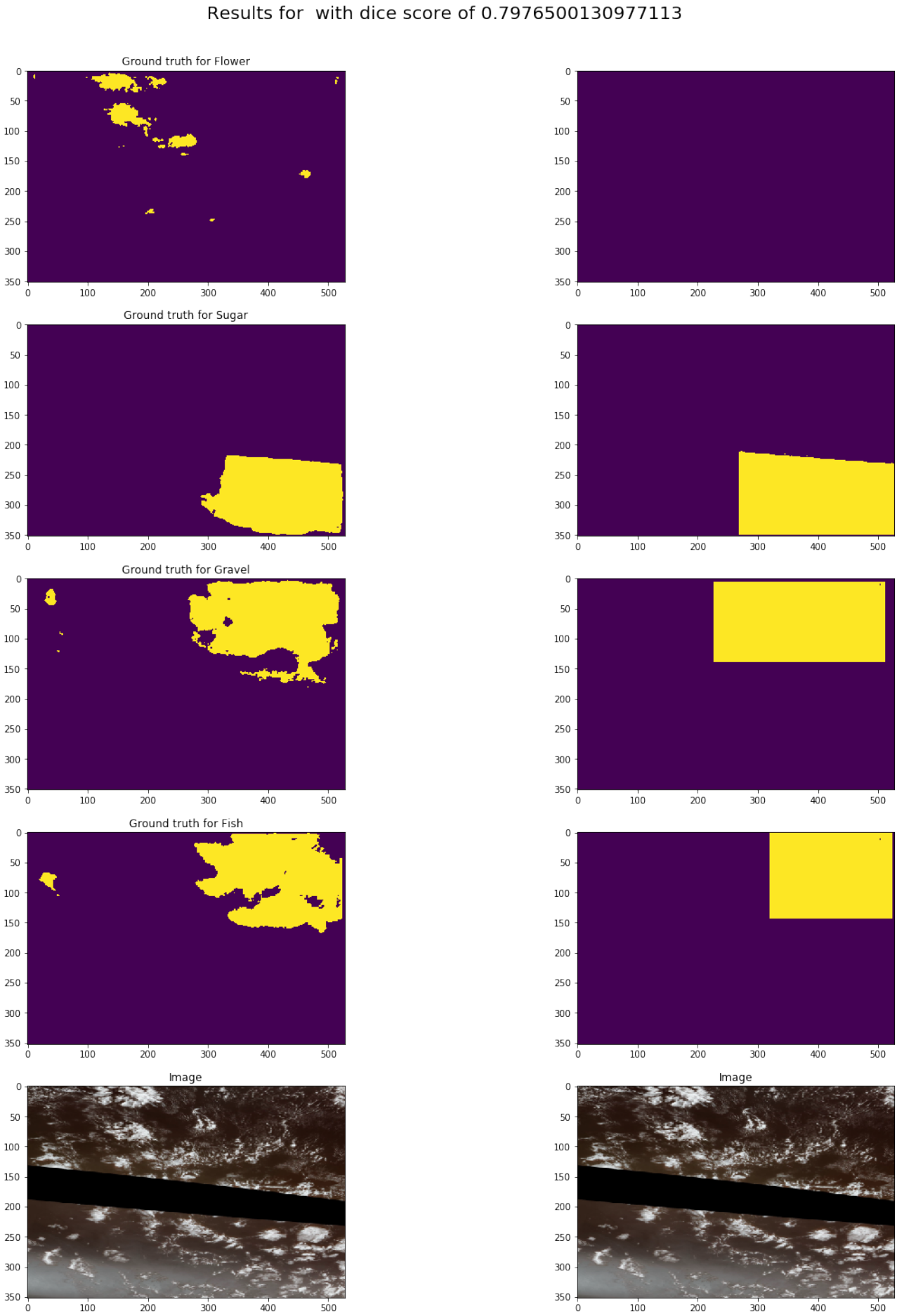

Figure 5: Visualization of best example from training set for U-Net.

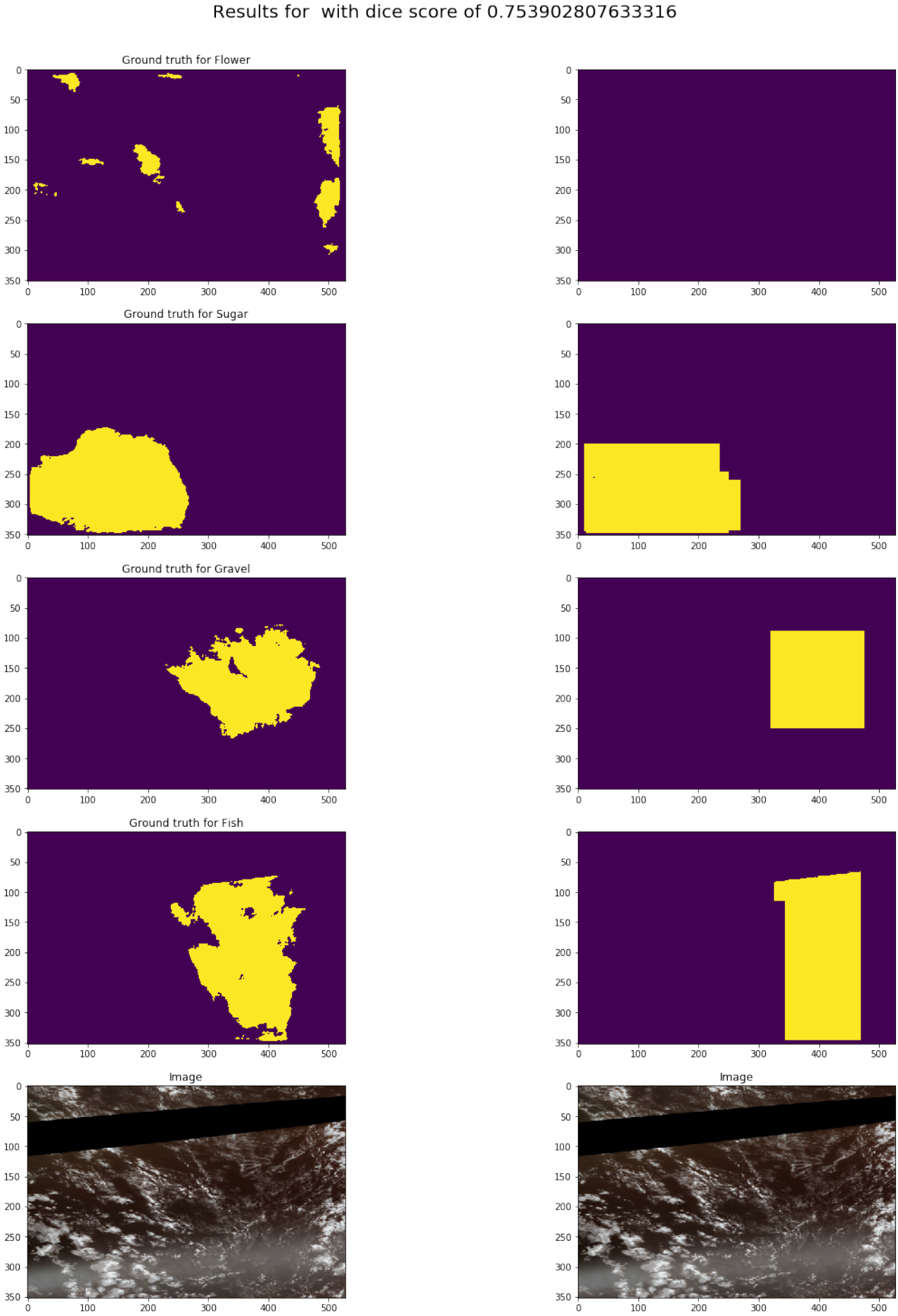

Figure 6: Visualization of best example from validation set for U-Net.

- 150 epochs with learning rate of $5 * 10^{-6}$ and identical parameters as mentioned above,
- 100 epochs with learning rate of $1 * 10^{-6}$, weight decay of 0.01 and rest of parameters unchanged.

Also, throughout the entire training we used batch sizes of 10 images for training and 2 for validation.

Performance of the model can be observed in Fig. 7 that depict the total loss for training and validation sets. One can observe that at first the validation curve tracked the training curve quite closely, yet, after 150 epochs it started to flatten out, while the training curve continued to diminish till 250 epochs when it also started to flatten. It may stem from the fact that the last run used twice as big weight decay parameter as previous runs to reduce the slight overfitting, yet the outcome was the opposite; namely the network did not generalize better.

Figures 8, 9 present the examples where the dice score was **the highest** for training (≈ 0.928) and validation (≈ 0.924) sets respectively. Also, for the whole datasets the network obtained dice coefficient of 0.393 for training set and 0.357 for validation. Such discrepancy between the results for the best examples and average for the entire dataset arises from the fact that the network performed poorly for some classes (namely Gravel). Comments of the possible improvement can be found in Sec.4.

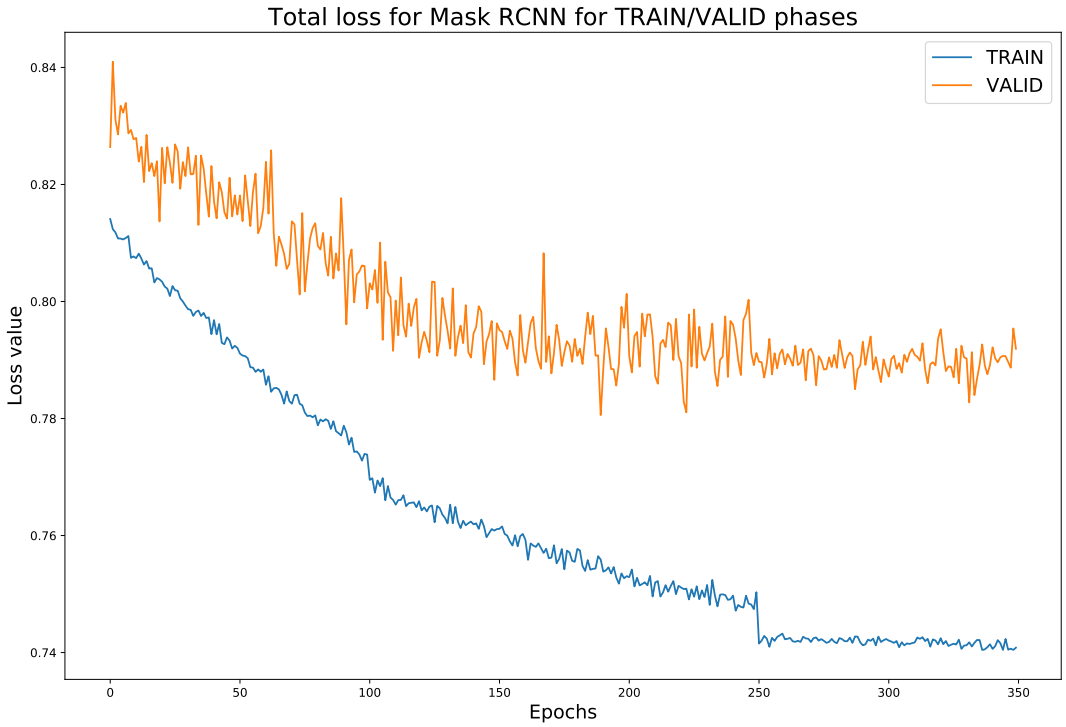

Figure 7: Loss of Mask R-CNN for training (blue) and validation (yellow) phases for 350 epochs.

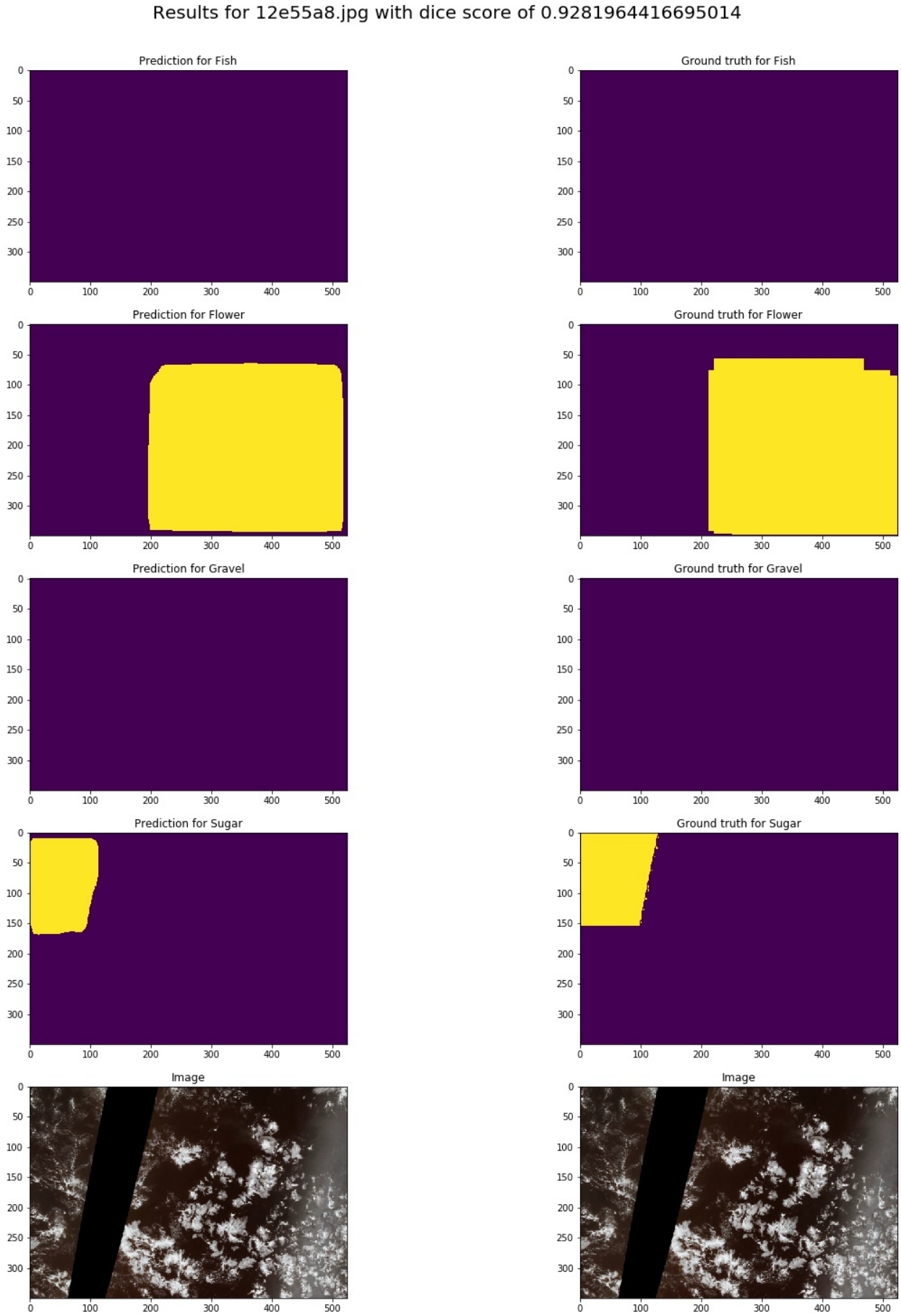

Figure 8: Visualization of best example from training set for Mask R-CNN.

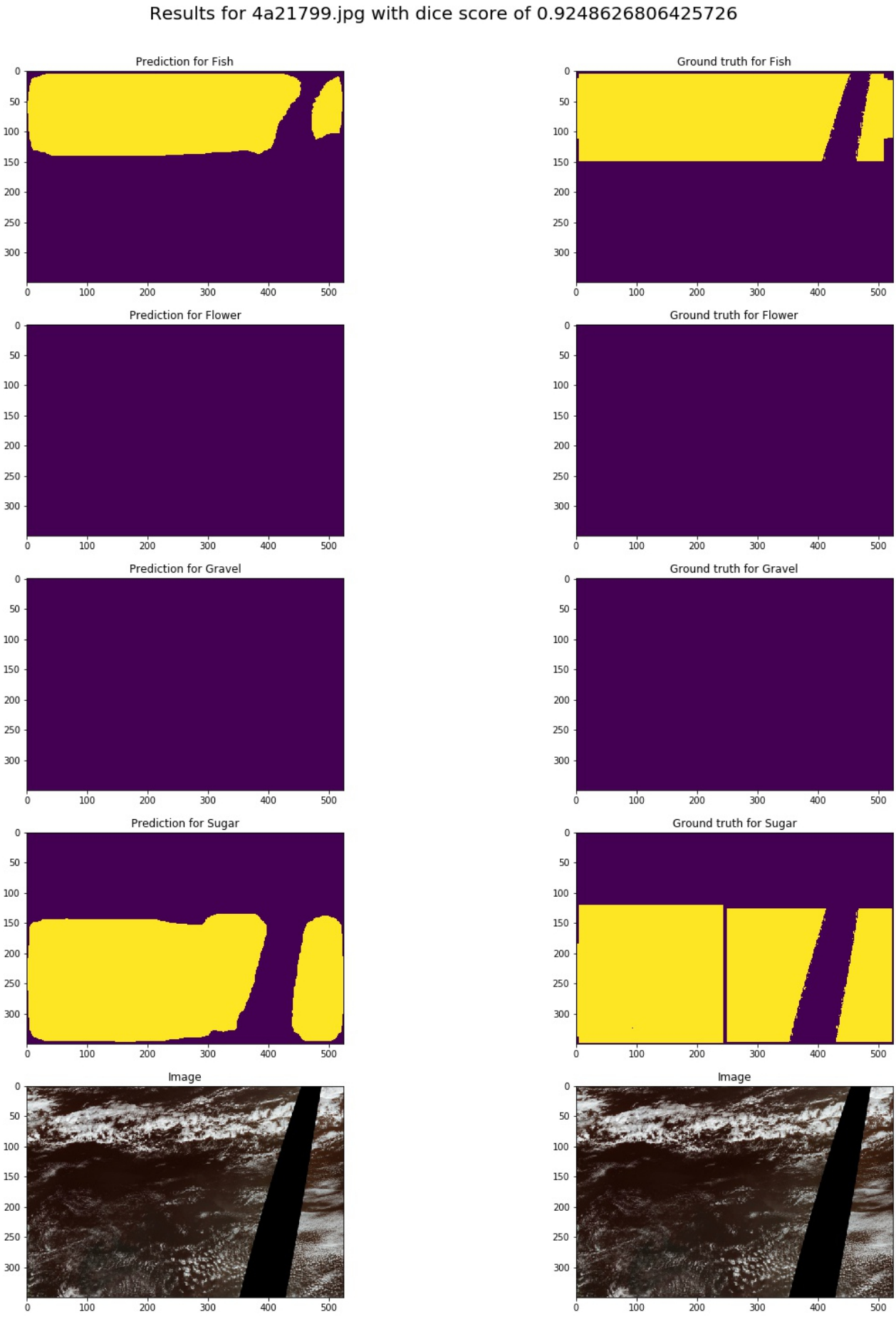

Figure 9: Visualization of best example from validation set for Mask R-CNN.

4 Summary and conclusions

The aim of this project was two-fold. Firstly, partaking in Kaggle competition required us to obtain the best possible results in the task of predicting masks (semantic segmentation) of satellite clouds images. We utilized two models, U-Net and Mask R-CNN with results presented in Tab.1, for the task of semantic segmentation. Mask R-CNN obtains higher results than U-Net which is expected as it is newer and more advanced neural network.

The networks exhibit overfitting (sever for U-Net, mild for Mask R-CNN) after half of the epochs passed, to address this issue the following methods might be applied:

- applying augmentations to the data, as they enlarge the dataset and help the model to discover more valuable patterns in the data. In the repository [2] there is a placeholder for the implementation of augmentations, yet during the development of this project we left it to be added in the future,
- abandoning the usage of weight decay as it did not provide any substantial gain,
- more careful way of choosing learning rate and learning rate decay (maybe exponential can be switched to multi-step learning rate decay)
- more experiments with the network hyperparameters
- changing the optimizer. We used the Adam [13] optimizer in both networks, we also tried SGD [14] optimizer with momentum but it provided significantly worse results. Yet, maybe another optimizer would be better.

Secondly, this was a university project, hence its aim was first and foremost to learn and get experience in a true machine learning project, so we spent a lot of time browsing through the articles, tutorials and code implementations of various solutions. Thus, even though the metrics are not as high as in the Kaggle leader-boards, we feel that it was well-spent time focused at broadening our horizons in machine learning fields.

Table 1: Dice score for two utilized networks (U-Net and Mask R-CNN) for entire training and validation sets

	U-Net	Mask R-CNN
Training set	0.244	0.393
Validation set	0.248	0.357

5 Work division

The work throughout the whole project was divide as follows (percentages):

- Maciej Kierkla - 20% for the dataset preparation part
- Jan Klamka - 20% for the U-Net part
- Wiktor Kondrusiewicz - 20% for the Mask R-CNN part
- Tomasz Bednarek - 20% for the analysis of the results of the models
- Marta Włodarczyk - 20% for making visualizations and writing the summary report

References

[1] https://www.kaggle.com/c/understanding_cloud_organization.

[2] https://github.com/wkondrusiewicz/understanding_clouds.

[3] https://pytorch.org/.

[4] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015.

[5] M. Buda, A. Saha, and M. A. Mazurowski, "Association of genomic subtypes of lower-grade gliomas with shape features automatically extracted by a deep learning algorithm," *Computers in Biology and Medicine*, vol. 109, p. 218–225, Jun 2019.

[6] https://github.com/mateuszbuda/brain-segmentation-pytorch.

[7] https://docs.opencv.org/master/d6/d00/tutorial_py_root.html.

[8] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," 2017.

[9] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[11] https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient.

[12] https://scikit-learn.org/stable/.

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[14] M. Li, T. Zhang, Y. Chen, and A. Smola, "Efficient mini-batch training for stochastic optimization," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 08 2014.