Chapter

Detecting and Counting Small Animal Species Using Drone Imagery by Applying Deep Learning

Ravi Sahu

Abstract

This work represents deep learning approach for detecting lizards on the summer grass background. It is the main part of general use case formulation—"how many animals are located now on this substitute habitat. Determine in which parts they prefer to stay". For this purpose, the U-Net architecture neural network was implemented. Dilated convolution layer was added to usual U-Net. Smoothly blending filter was applied to result probability patches for connecting them in one big probability map without sewed edges. Designed flexible architecture allows to train neural network for pixel-wise semantic segmentation with accuracy value 0.9863 on the tiny dataset.

Keywords: machine learning, deep learning, semantic segmentation, U-Net, Keras, blending filter

1. Introduction

In 2018, the strictly protected sand lizards were relocated from several construction sites to this formerly military-used area. If possible, even more animals should be relocated here in the following years. Special habitat elements were created for this area. The fenced area is grazed, so that there is no need for mowing.

The conservation status is controlled by monitoring the population. For this purpose the animals have to be counted rotationally.

Machine learning (and deep learning in particular) focused on using modern mathematics and programming tools to figure out numerical presentation of abstract and stochastic source data. In the past few years, deep learning helped moving the computer vision to production by extending possibilities and improving result accuracy.

Before starting the model synthesis for the task of lizard localization on the ground, we have collected objective features (issues):

- The object is very small. To be able to distinguish from leaves and branches, drone camera should detect a back drawing of lizard and its legs.

- Distribution of lizards on the ground is very small. After reviewing 1800 sample, images of two to three lizards were found.

- Nature has created lizards very similar to the leaves around them (natural camouflage effect).

- Partially displayed. Except two to three lizards which were found in source dataset, some potential objects were found too. It could be lizards hidden under branches, leaves, and grass.

- Different sunlight angle and brightness.

- Motion blurring.

Some of these issues could be delegated to drone-shooting side, but most parts should be solved by machine learning.

2. Data

Running ahead we could say that it is a classical binary classification problem where the prediction result will be measured in continuous space.

In the notation of binary classification problem, there are two classes: positive—part of image where lizards exist; and negative—all except lizards.

To narrow the negative class presentation, and make it representative for our task, negative samples for this class were taken from the source image dataset. In such a way, we specify all features on the ground except lizards as a negative class (**Figure 1**).

The positive class was represented by extremely small number of lizards from the initial dataset.

We reviewed manually 1800 images from the drone dataset, and only two to three lizards had been found. It appeared that we have not enough positive images for training. To solve this problem and move forward, we reused lizards' images

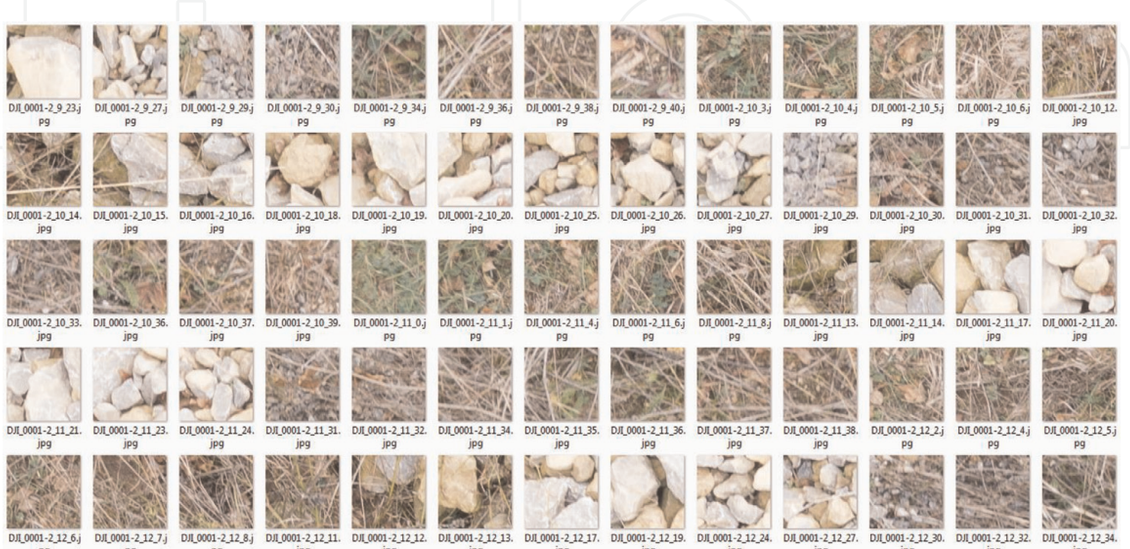

Figure 1.
Example of negative sample images.

Figure 2.
Example of positive sample images.

from the Internet. About 1000 lizard images were taken from "ImageNet" dataset. After obvious filtering, about 600 images remained. We left green lizards which were shot from the top view (similar to drone view dataset) (**Figure 2**).

For labeling masks for positive image samples, we used simple and standalone manual annotation software "VGG Image Annotator (VIA)" [1].

At the end of data presentation, we should note about the great approach of growing the source dataset. The examples available for learning were limited so the classification problem added a layer of complexity. So this is a challenging machine learning problem, but it is also a realistic one: in a lot of real-world use cases, even small-scale data collection can be extremely expensive or sometimes near-impossible. Being able to make the most out of very little data is a key skill of a competent data scientist.

In order to make the most of our few training examples, we "augment" them via a number of random transformations, so that our model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better:

- Flip vertical/horizontal
- Rotation
- Translation
- Zooming
- Color desaturation

3. Training scheme

Two-steps training scheme separates "rough training" and "subtle training". Each step is supported by fourfold cross-validation to obtain the best representative results. Well parameterized model allows using the same model with different parameters (**Figure 3**):

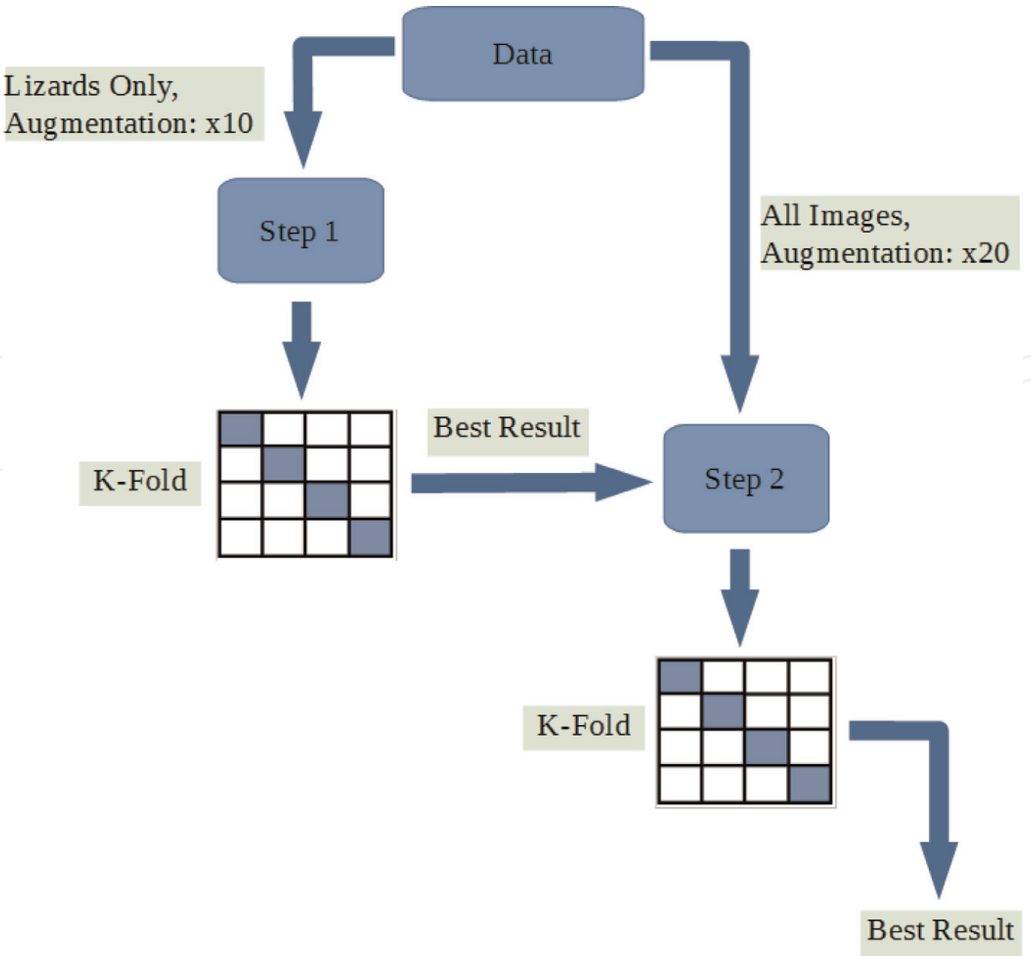

Figure 3.
Multistage scheme of the training process.

Step 1 (rough training)
Aims:

- Avoid class imbalance during initial training
- Attention to local minima

Methods:

- Training on samples which contains positive objects only (lizards)
- Rough regularization to avoid overfit
- Augmentation increase dataset size by ×10
- Optimizer: Adam (learning rate: 0.001)

Step 2 (subtle training)
Aims:

- Training to obtain the best possible result
- Attention to global minima

Methods:

- Training on all samples (as lizards as background)
- Tiny regularization
- Augmentation increase dataset size by ×20
- Optimizer: Nadam (learning rate: default)

4. Data generator

Parameterized data generator allows to control the data filing used for training and for testing. Augmentation allows to drastically increase the dataset size by image transforming. Content filter allows filling the control of the percent of positive (lizards) pixels in each image used for training (**Figure 4**):

- Image augmentation parameters
  - Rotation up to 360 degrees
  - Width/height shifting up to 20%
  - Shear up to 20 degrees
  - Zoom +/− up to 20%
  - Random horizontal/vertical flip
  - Fill mode: "reflect"
- Content data filter controls class-imbalance in source dataset
- K-fold splitter provides datasets for separate training and implement cross-validation

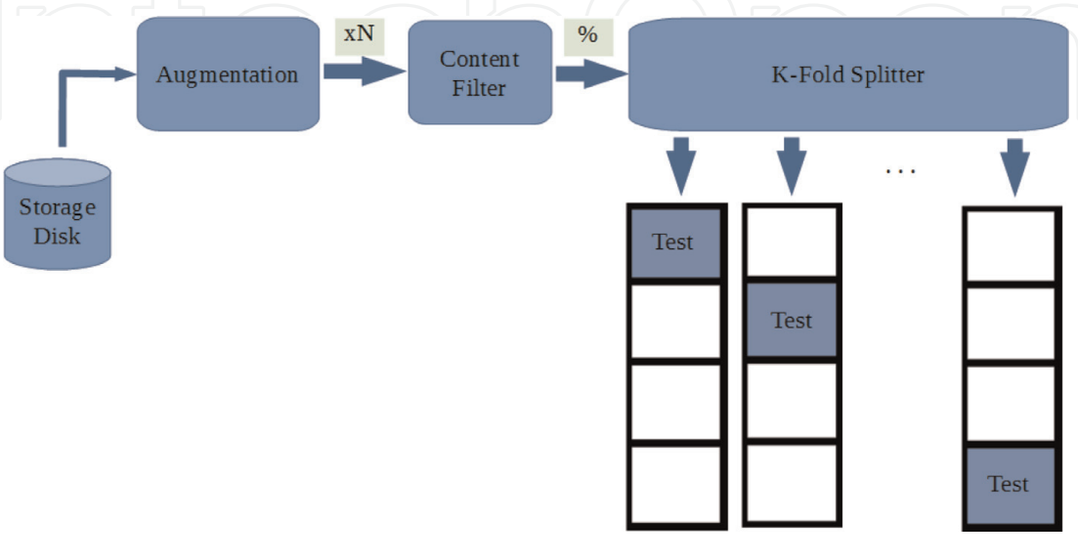

Figure 4.
Scheme of data generator.

5. Model

We did not reuse the existing solution like YOLO [2] or MASK-RCNN [3] because benefits of these networks focused on relatively big objects with at least 10–20% covered by a whole image. In case with lizards, we have only 0.08% coverage of the object on the image.

Instead, the model architecture was based on the well-known pixel-wise image segmentation approach "U-Net" [4] and extended by adding atrous/dilated convolution layers [5] to the low-resolution part of the neural network.

The model was implemented in Keras [6]—a high-level neural network API, written in Python and capable of running on TensorFlow, CNTK, or Theano.

Input layer "input_1" gets three-channel (RGB) image in range 0–255.

Pre-processing layer "lambda_1" converts input 255-ranged values to 0–1 range.

As the usual U-Net neural network, it has encoder and decoder parts, connected each other via residual connections.

Encoder block contains a pair of convolution layers with regularization block in the middle and max pooling layer in output (**Figure 5**).

We found that encoder block requires only dropout regularization.

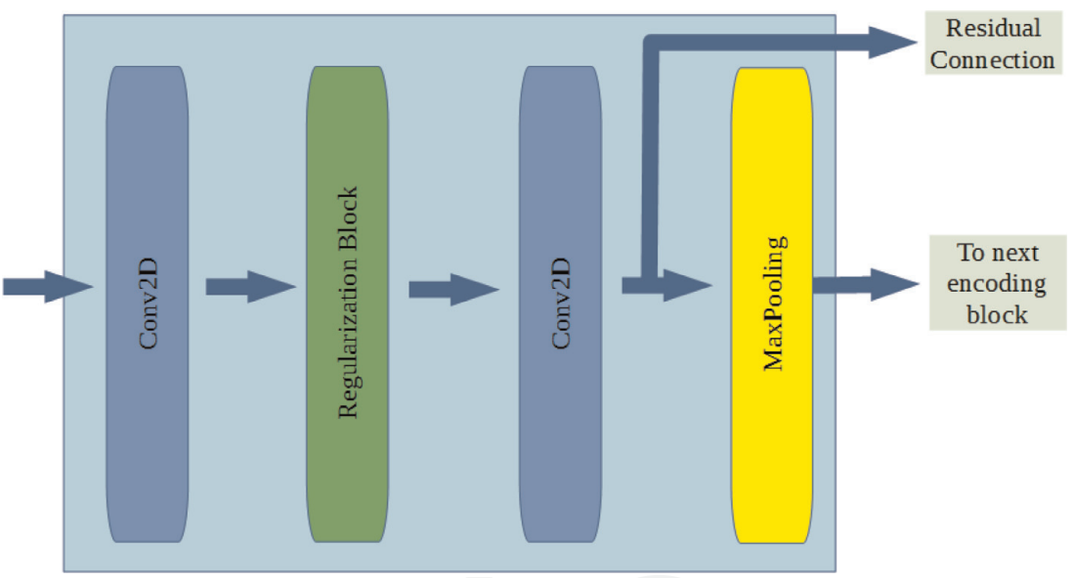

Figure 5.
Encoding block.

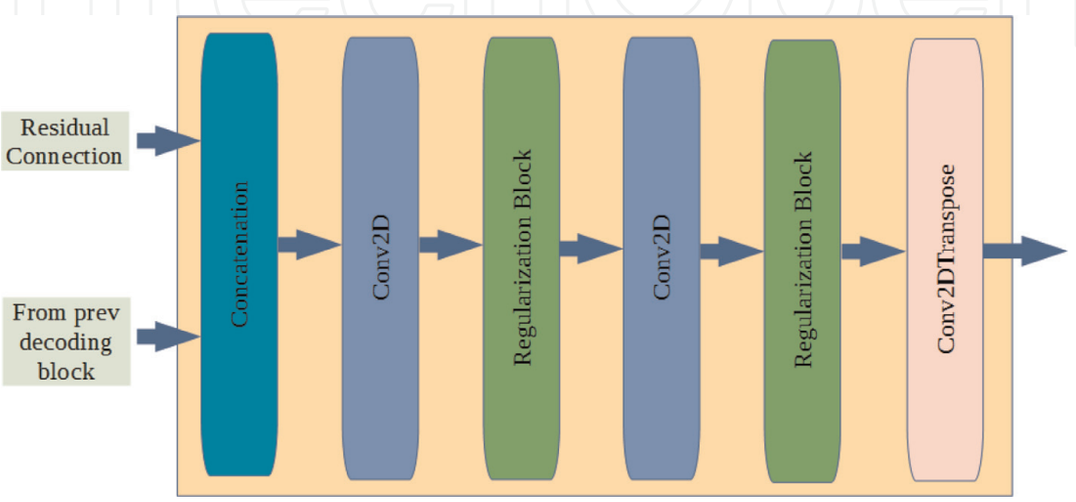

Figure 6.
Decoding block.

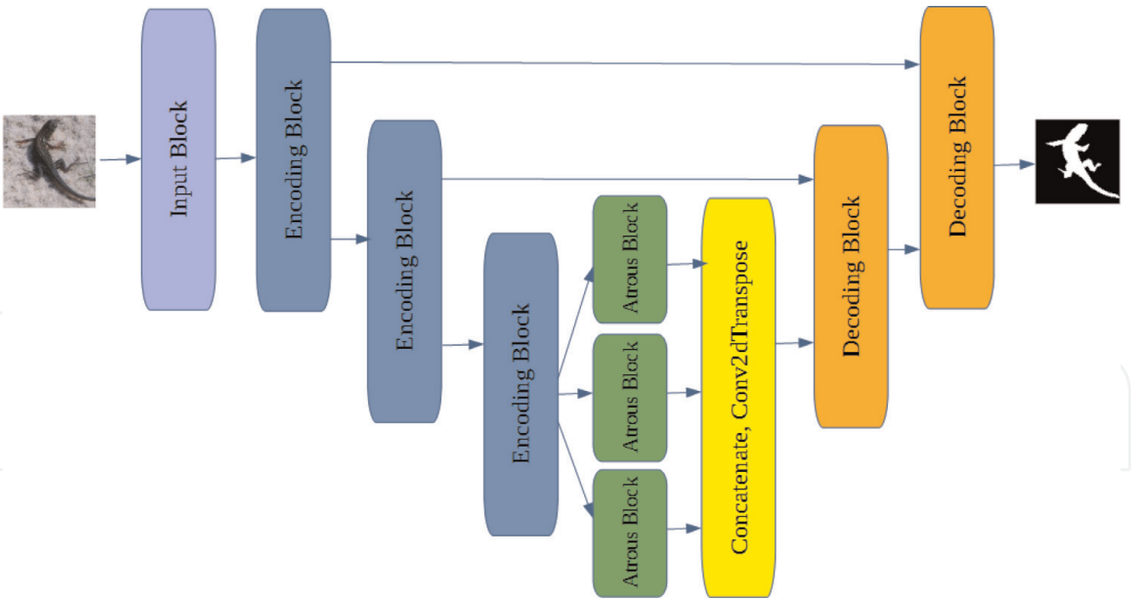

Figure 7.
Final scheme of neural network model.

Decoder block contains a pair of convolution layers with regularization block in the middle as well as encoder block, but instead of max pooling output layer, it uses Conv2DTranspose layer and input layer concatenation to connect the result of the previous block with residual connection (**Figure 6**).

We found out the best regularization for decoder block is mix dropout with batch normalization.

To extend basic U-Net, we added three dilated convolution blocks with dilation rates 4, 8, and 12 between encoder and decoder parts of the NN.

The final block scheme of used neural network is displayed in **Figure 7**.

Model details:

- Encoder/decoder convolution layers
    - Initializes: He normal
    - Padding: Same
    - Kernel size: 3
    - Activation function: Rectified linear unit (ReLU)
- Dilated convolution layers
    - Initializes: He normal
    - Padding: Same
    - Kernel size: 1
    - Activation function: Rectified linear unit (ReLU)
    - Dilation rates: 4, 8, 12

- Max Pooling
    - Pooling size: 2
- Conv2DTranspose layers (except output layer)
    - Initializes: He normal
    - Padding: Same
    - Kernel size: 3
    - Activation function: Rectified linear unit (ReLU)
    - Strides: 2
- Conv2DTranspose layers (output layer)
    - Initializes: Glorot/Xavier normal
    - Padding: Same
    - Kernel size: 3
    - Activation function: Sigmoid
    - Filters: 1
- Loss function: BinCrossE-Log(Jaccard index)

It is obvious for the binary classification task to use logistic sigmoid function [7] as activation function in output layer which represents the probability of relation between each pixel to positive class.

Loss function of the neural network was binary cross entropy [8] extended by Jaccard index [9].

Winners of Kaggle competitions, who used U-Net for proving their approach said:

"It is well known that in order to get better results your evaluation metric and your loss function need to be as similar as possible. The problem here however is that Jaccard index is not differentiable. One can generalize it for probability prediction, which on one hand, in the limit of the very confident predictions, turns into normal Jaccard and on the other hand is differentiable—allowing the usage of it in the algorithms that are optimized with gradient descent" [10].

6. Post-processing

In fact, the U-Net takes an image patch and makes predictions on those small local windows, without data near the border of the patches, so there might first be a high error on the predictions made near the boundary of the window, in plus of the fact that predictions may be just concatenated, so it looks even more jagged (**Figure 8**).

To get with this problem, we reused well-designed solution prepared to deal with it—"blending predicted patches smoothly is a must to please the human eye."

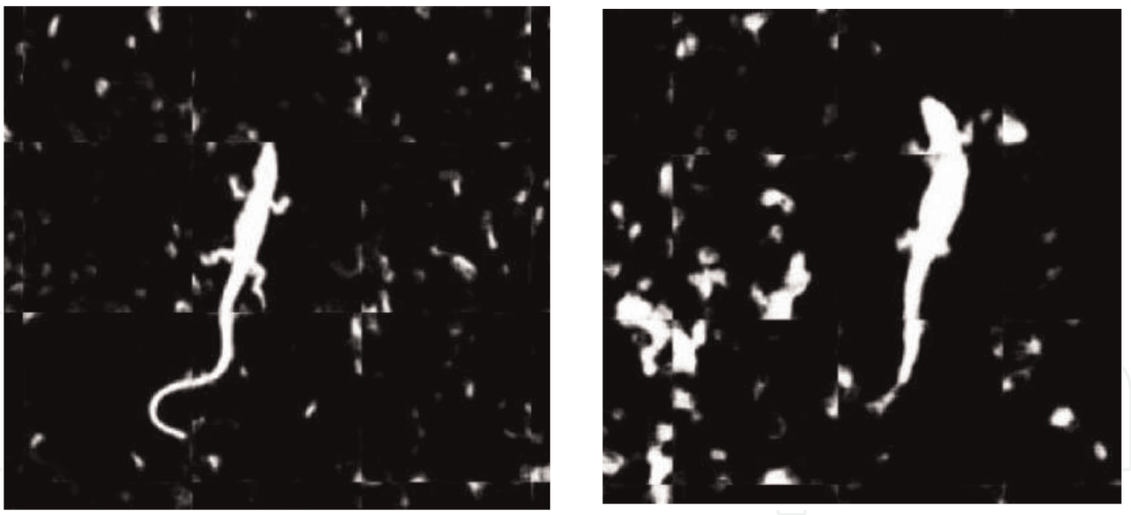

Figure 8.
Connected patches without any blending.

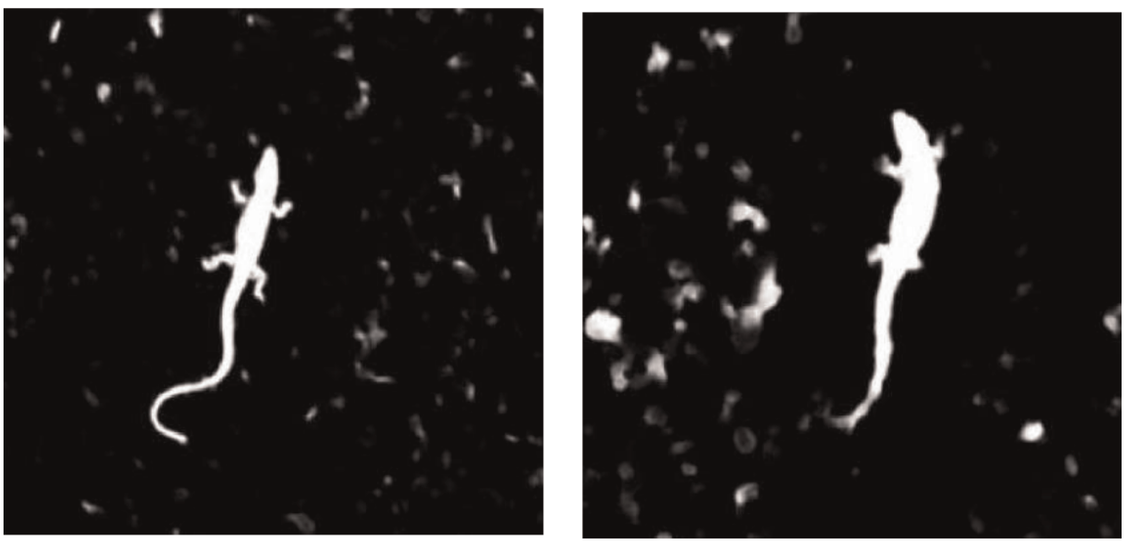

Figure 9.
Connected patches with applied blending.

(https://github.com/Vooban/Smoothly-Blend-Image-Patches)
To get smoothed results, the following steps are applied to each patch:

- Use the four possible 90 degrees rotations, as well as a mirrored version of those rotations, so as to augment the images eightfold for prediction before blending the predictions together

- 2D interpolation between overlapping patches when doing the final predictions

After applying the post-processing filter, we obtain the following probability map for the same samples (**Figure 9**).

7. Logs

We used online service https://www.comet.ml for monitoring the training process. Following are the results of training network with different hyperparameters values (**Figures 10–11**).

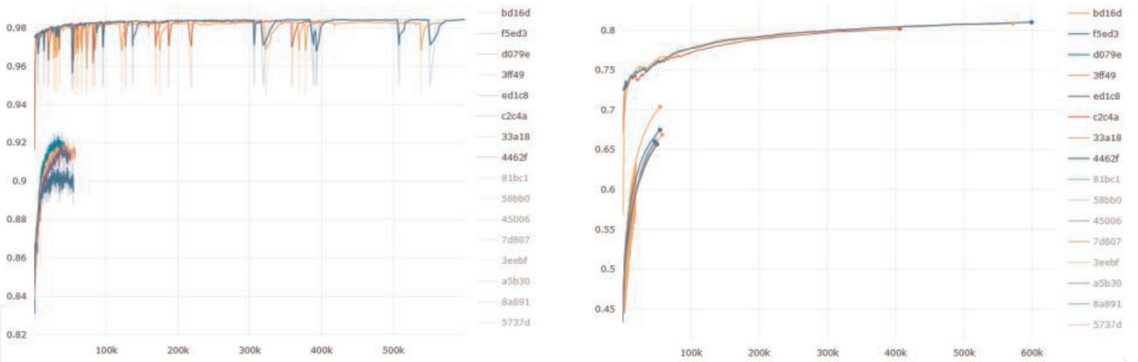

Figure 10.
The model's training history which does not use bias in convolution layers and does not use additional batch normalization in atrous blocks.

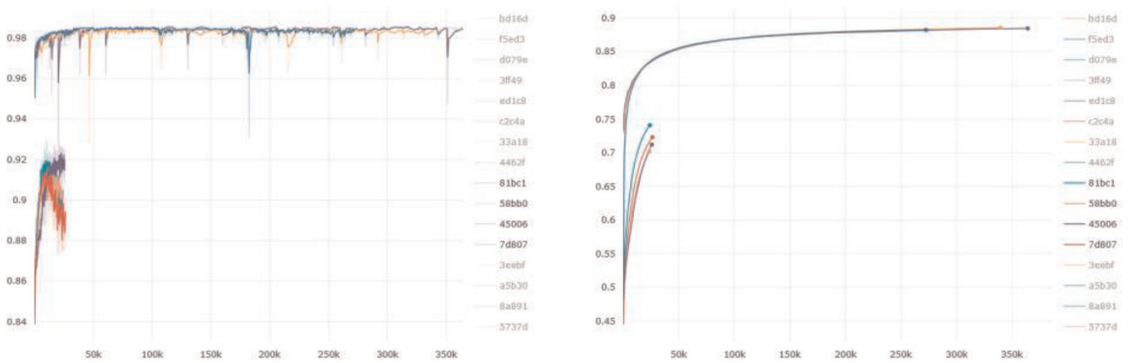

Figure 11.
The model's training history which uses bias in convolution layers and does not use additional batch normalization in atrous blocks.

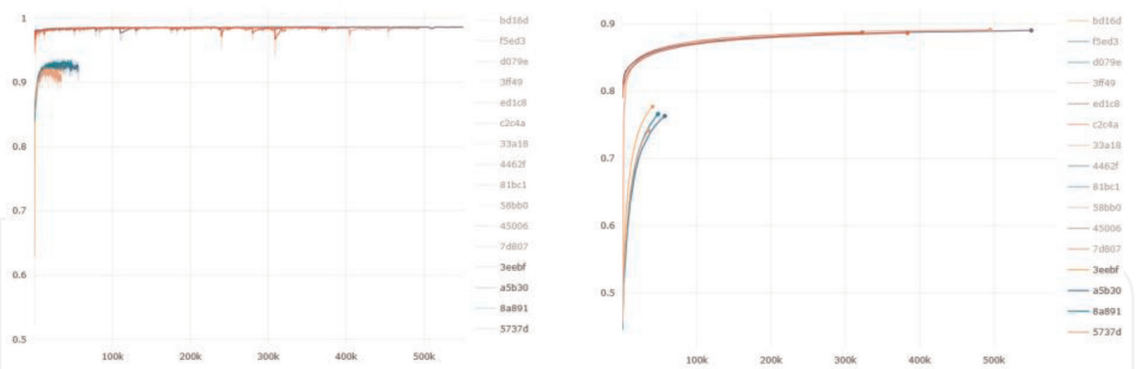

Figure 12.
Model's training history which does not use bias in convolution layers and use additional batch normalization in atrous blocks.

7.1 Model does not use bias in convolution layers and does not use additional batch normalization in atrous blocks

Best validation accuracy: 0.9841.
Best Jaccard index: 0.8098.

7.2 Model use bias in convolution layers and do not use additional batch normalization in atrous blocks

Best validation accuracy: 0.9844.
Best Jaccard index: 0.8857.

7.3 Model does not use bias in convolution layers and use additional batch normalization in atrous blocks

Best validation accuracy: 0.9863.
Best Jaccard index: 0.8913.

8. Results

Below are the set of result images taken from a drone with highlighted objects predicted as lizards.

The color of the highlighted rectangles has a range from blue to red with respect to the probability of the predicted object. Blue rectangles highlighted predictions with low probability. Red rectangles highlighted predictions with high probability (**Figures 13–15**).

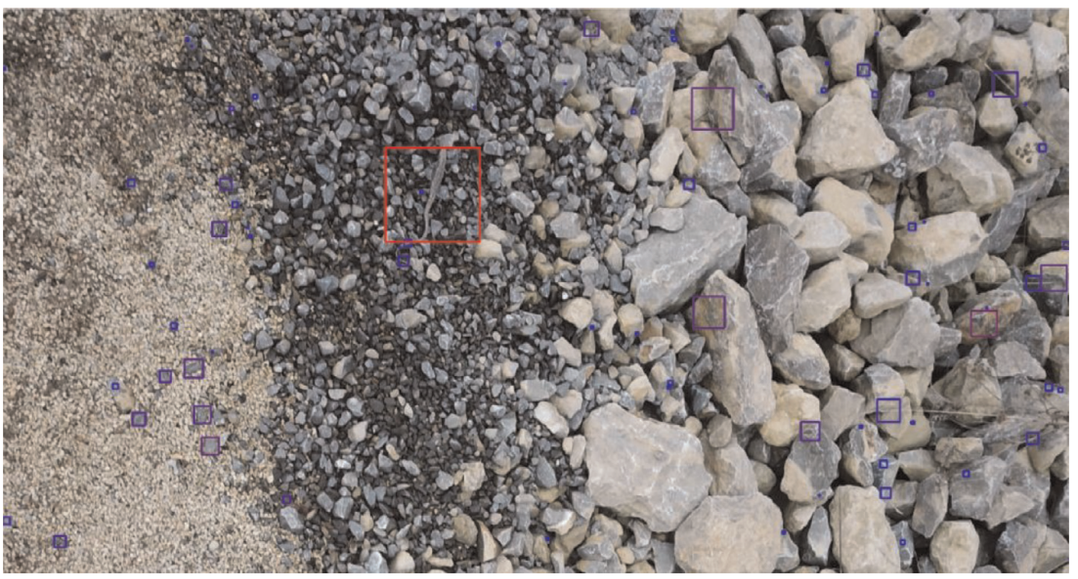

Figure 13.
Example of detection. Gray lizard detected with high probability. There are gaps between rocks that could have quite high probability too.

Figure 14.
Example of detection. Green lizard detected with high probability. Branches have less but detected probabilities.

Figure 15.
Example of detection. One gray (between rocks) and one white (on the rock) lizard detected with high probability. Branches have less but detected probabilities.

9. Conclusion

The design and the architecture of the developed application allowed us to build a flexible model of the neural network and find the best configuration of hyperparameters.

As was found with the help of using CometML service, the best case of hyperparameters is:

"Model does not use bias in convolution layers and use additional batch normalization in atrous blocks."

Best validation accuracy: 0.9863.

Best Jaccard index: 0.8913.

The following is a list of implemented features:

- Disk-caching allows to use unlimited size of datasets and avoid memory overflow problem.
- Used GPU memory amount control allows using several trainings on the same GPU.
- Attention to reproducing trainings allows to compare different results.
- Source data augmentation allows training on the tiny-size dataset.
- K-fold cross-validation allows training more deeper and obtain better results.
- Parametrized training allows stacking training solutions.
- Structured solutions allows to build multistep training.
- Post-processing allows to obtain smoothly blended probability for the whole image and, as a result, better prediction quality for big image resolution.

Author details

Ravi Sahu
Strayos, St. Louis, MO, USA

*Address all correspondence to: ravi@strayos.com

References

[1] Dutta A. Visual Geometry Group. Oxford University. Available from: http://www.robots.ox.ac.uk/~vgg/software/via/

[2] Redmon J, Divvala SK, Girshick RB, Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. Available from: https://arxiv.org/abs/1506.02640

[3] He K, Gkioxari G, Dollár P, Girshick G. Mask {R-CNN}. Available from: https://arxiv.org/abs/1703.06870

[4] Ronneberger O, Fischer P, Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. Available from: https://arxiv.org/abs/1505.04597

[5] Chen L-C, Papandreou G, Schroff F, Adam H. Rethinking Atrous Convolution for Semantic Image Segmentation. Available from: https://arxiv.org/abs/1706.05587

[6] Chollet F. Available from: https://keras.io/

[7] https://www.sciencedirect.com/topics/computer-science/sigmoid-function

[8] http://neuralnetworksanddeeplearning.com/chap3.html

[9] https://www.sciencedirect.com/topics/computer-science/jaccard-coefficient

[10] Iglovikov V, Mushinskiy S. Kaggle: Dstl Satellite Imagery Competition, 3rd Place Winners' Interview: Vladimir & Sergey. Available from: http://blog.kaggle.com/2017/05/09/dstl-satellite-imagery-competition-3rd-place-winners-interview-vladimir-sergey/