The Higgs boson machine learning challenge

Bruno Magalhaes[1], Riccardo Silini2

Abstract
bla bla bla

Email (SCIPER): [1] bruno.magalhaes@epfl.ch (212079) [2] riccardo.silini@epfl.ch (214398)

1. Introduction

2. Data Augmentation

1. Rotation of input images with non-orthogonal roads We increased the number of input images from 100 to 148 by rotating 90deg, 180deg and 270deg the (16) training images that are not composed only of non-horizontal and non-vertical roads. This feature is activated by the flag `DATA_AUGMENTATION`.

2. Patches from concatenation points We increase by almost two-fold the number of patches per training image by performing extraction of all patches of size `IMG_PATCH_SIZE` that are placed at every `IMG_PATCH_SIZE/2` interval. This feature is activated by the flag `ADD_INTERCALATED_PATCHES`. In practice, it extracts patches with the corners at the mid-point positions of previously extracted patches, as displayed below:

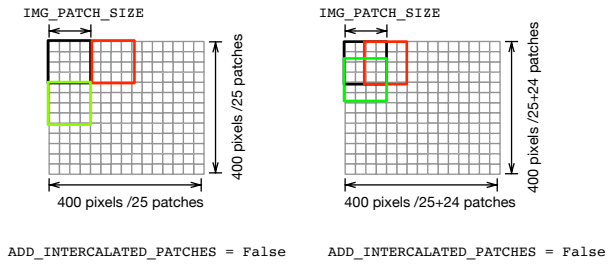

Figure 1. The collection of patches by stepping `IMG_PATCH_SIZE` (default, left) compared with `IMG_PATCH_SIZE/2` (`ADD_INTERCALATED_PATCHES` activated, right). When traversing image, next collected patch on the horizontal (vertical) direction is displayed as red (green).

3. Features

Two-step classification To reduce noise and allows classification at a finer scale, we added a second neural network to the processing. The initial one will be trained with pairs of [images,groundtruth] and outputs the segmented classification. The second network is trained with pairs [classification,groundtruth] and received the first network's classification as input, outputting the final classification. The second step of this operation is presented as `Phase 2` in the source code. An example of an application of the two-step classification is displayed in the following picture.

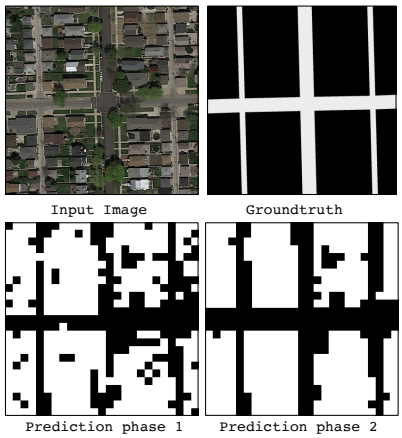

Figure 2. An example of the application of the two-phase propagation process: for a given input (top-left), the predicted segmentation at output of the neural network (bottom-left) is then re-input to a second neural network trained from groundtruth images (top-right), outputting the final classification (bottom-right).

Analysis of neighborhood information This functionality can be activated wit the flag `NEIGHBORHOOD_ANALYSIS` set to `True` and definining the `NEIGHBOR_PIXELS` to the margin of neighborhood pixels to be added and analysed from each patch.

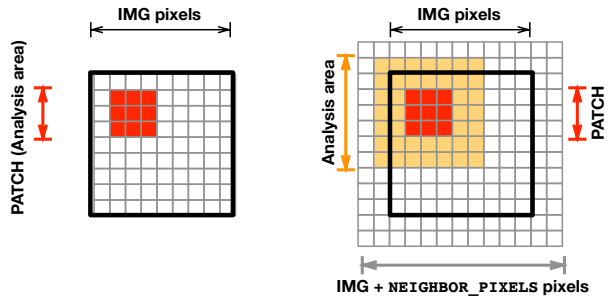

Figure 3. Example of the neighborhood analysis feature. Standard classification (left) inputs patches of images and classifies the same area. With neighborhood analysis (right), we input the patch of size `IMG_PATCH_SIZE` and all surrounding `NEIGHBOR_PIXELS` pixels (orange), with the learning based on the labels of the patch only (red).

Randomized input sequence The order in which we present the observations (input vectors) comprising the training set to the network affects the final computation of weights. We implemented a randomized arrangement of the observations according to the response variable that has been shows to be present a better tuned network when compared with ordered arrangements (the standard implementation). Our input is then a randomized set of classification with balanced number of inputs for each label. This feature is enabled by defining `RANDOMIZE_INPUT_PATCHES = True`.

Dropout Dropout is a regularization technique for reducing the overfitting of neural networks. It performs model averaging of neural networks efficiently and prevents complex co-adaptations on training data (Srivastava et at. 2014 [1]). We enabled dropout in our models by specifying the percentage of dropout on the variable `DROPOUT_RATE` as exemplified in the next figure:

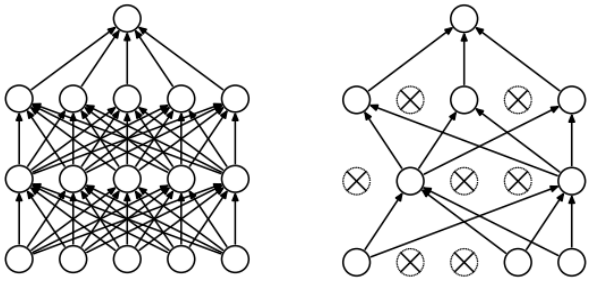

Figure 4. Example of the dropout feature. Standard weights tunning (left) methods apply an update of all weights. Dropout-based networks (right) discards the updates of certain weights with the rationale of reducing overfitting. In the example, a dropout rate of `0.5` disables half of the nodes on the network.

Multi-Layer convolutional network We allow the sub-network of convolution layers to increase or decrease, according to the user-defined parameters `CONV_LAYERS`. This allowed us to play with the quality of model fitting for different set set of parameters. In practice, our network is an array of convolution and pooling layers, where the output of one layer is the input of the following one. The parameters of every layer is fully customisable by the user from the following variables:

- `CONV_LAYERS`: number of convolutional layers, and size of the following parameter arrays;
- `CONV_FILTER_SIZES = [5,5,5,5]` : For each layer, the count of horizontal and vertical pixels of filter;
- `CONV_FILTER_DEPTHS = [32, 64, 128, 256]`: depth of the weights for the aforementioned layers;
- `POOL_FILTER_STRIDES = [2, 2, 2, 2]`: horizontal and vertical stride for the max-pooling on each layer;
- `FC1_WEIGHTS_DEPTH = 512`: depth of the first fully connected layer, connected to conv. layer of id `CONV_LAYERS-1`. The second layer performs binary probability classification therefore has depth `NUM_LABELS` (road; non-road);

This allowed us to play with the quality of fitting based on user-provided filter sizes, depths and strides – as detailed in the following paragraph.

Final Network Architecture In brief, our network is composed by a set of convolution layers (for extracting of convoluted features) and pooling layers (for the extraction of most meaningful weights from convolutional layer outputs), followed by two layers of fully connected layers (the universal function approximators). The input is a set of images to which patches are extracted. Those patches are then input individually or in batches to the network. The output contains the probability of the road or non-road classification. A sample of a possible network architecture with 2 convoluted layers and default parameters is displayed in figure 5.

4. Performance Analysis and Results

Riccardo bla bla bla

5. Final Remarks and Conclusion

bla bla bla

References

[1] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

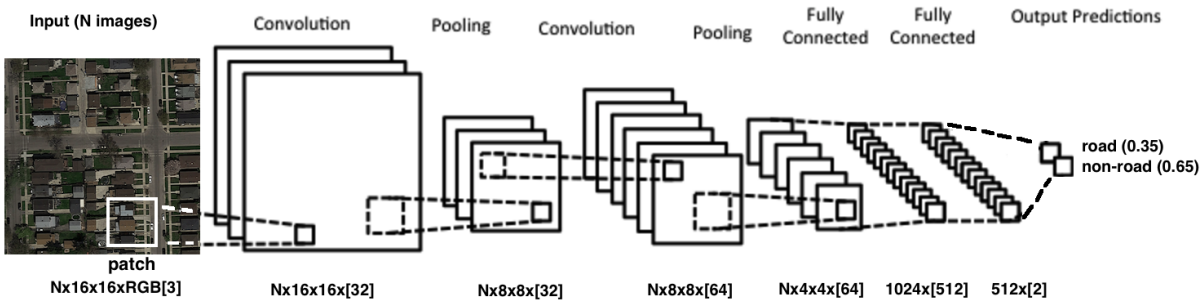

Figure 5. A general overview of the deep network architecture, with default parameters. Dimensionalities, padding types, depths and number of convolution+pooling layers are user-customisable.