

Anomaly Detection for Crop Monitoring

Stefano Cerri

Politecnico di Milano

stefano1.cerri@mail.polimi.it

June 2, 2016

Overview

- 1 Introduction
- 2 Extract Patches
- 3 Train the network
- 4 Evaluate the network
- 5 Summary and future extensions

One of the most important tasks to be addressed in intelligent farms is the crop monitoring. Computer Vision and Machine Learning algorithms are typically combined to analyze and classify images acquired by autonomous robots that monitor the growth of crops, and in particular, to discriminate crops (i.e. the good plants) from weeds (any undesirable or unwanted plant growing wild, especially those that takes food or nourishment from crops). Crop/weed discrimination are often tacked as an image-classification problem.

The dataset used in this project is the CWFID dataset¹. It comprises field images, vegetation segmentation masks and crop/weed plant type annotations. The paper details, e.g. on the field setting, acquisition conditions, image and ground truth data format. All images were acquired with the autonomous field robot BoniRob in an organic carrot farm while the carrot plants were in early true leaf growth stage.

¹<https://github.com/cwfid/dataset>

Assignment

The assignment is to create a Convolutional Neural Network, using deep learning, that classifies the images in crop, weed or ground.

This work was done by two colleagues, Jorge Carpio Lopez de Castro and Andrea Luigi Edoardo Caielli, that work on the same project. The result of the script is a dataset composed of:

- **3000 training patches** : 1000 crop, 1000 weed, 1000 ground
- **3000 testing patches** : 1000 crop, 1000 weed, 1000 ground

Example of Patches

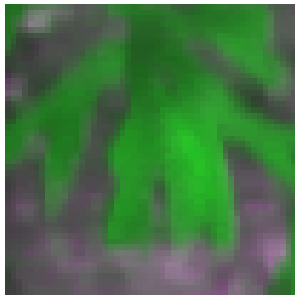


Figure: crop patch

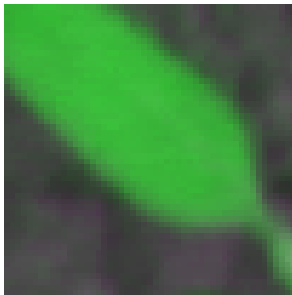


Figure: weed patch

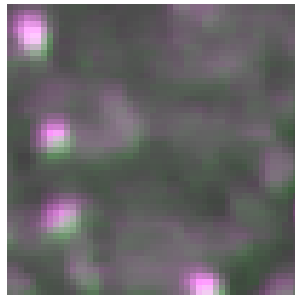




Figure: ground patch

To perform the given assignment I downloaded and installed MatConvNet library². Note that in order to use some functionalities of the library, such as using GPU acceleration, it has some requirements³. There is also a manual⁴ that explains all the blocks of a CNN both from the "mathematical" and "code function" view

²<http://www.vlfeat.org/matconvnet/>

³<http://www.vlfeat.org/matconvnet/gpu/>

⁴<http://www.vlfeat.org/matconvnet/matconvnet-manual.pdf>  

Example of CNN

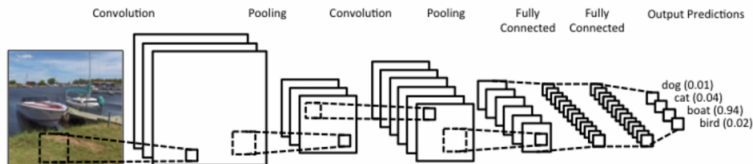


Figure: Example of a Convolutional Neural Network

The typical layer of a Convolutional Neural Network are:

- **Convolutional layer**
- **Pooling Layer**
- **Fully-connected layer**
- **ReLu**
- **SoftMax**

Selected Architectures

I selected different network architectures for comparing it, after the training, with the testing set. The architectures are:

- ① $INPUT \rightarrow [CONV \rightarrow RELU \rightarrow POOL] * 2 \rightarrow FC \rightarrow RELU \rightarrow FC \rightarrow LOSS$
- ② $INPUT \rightarrow [CONV \rightarrow RELU \rightarrow POOL] * 3 \rightarrow FC \rightarrow LOSS$
- ③ $INPUT \rightarrow [CONV \rightarrow RELU \rightarrow POOL] * 4 \rightarrow FC \rightarrow LOSS$
- ④ $INPUT \rightarrow [CONV \rightarrow RELU \rightarrow POOL] * 3 \rightarrow FC \rightarrow LOSS$
- ⑤ $INPUT \rightarrow [CONV \rightarrow RELU \rightarrow POOL] * 3 \rightarrow FC \rightarrow RELU \rightarrow FC \rightarrow LOSS$
- ⑥ $INPUT \rightarrow [CONV \rightarrow RELU \rightarrow POOL] * 3 \rightarrow [FC \rightarrow RELU] * 2 \rightarrow FC \rightarrow LOSS$

Results of the training

Here are the 6 networks training graph. On the left side there is the objective function and on the right side there is the error function.

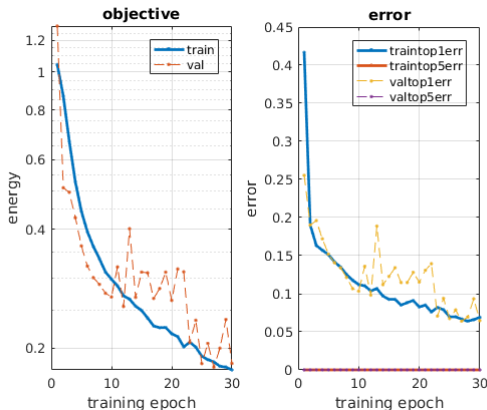


Figure: network 1 training

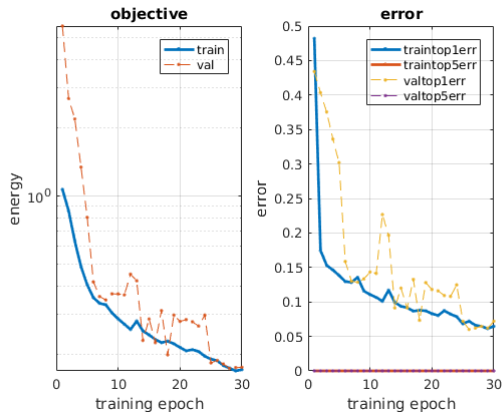


Figure: network 2 training

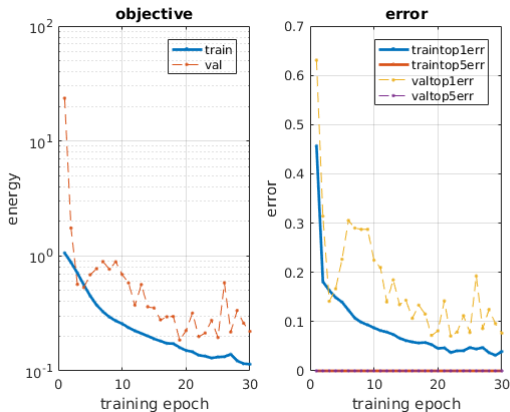


Figure: network 3 training

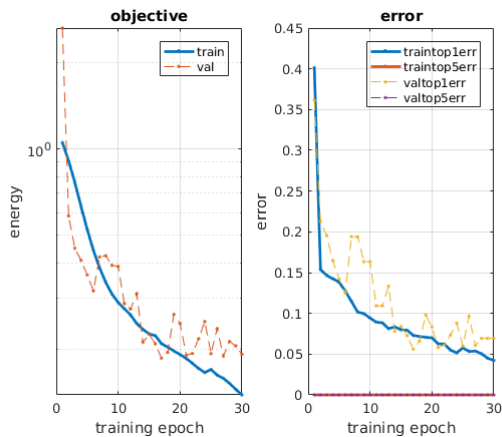


Figure: network 4 training

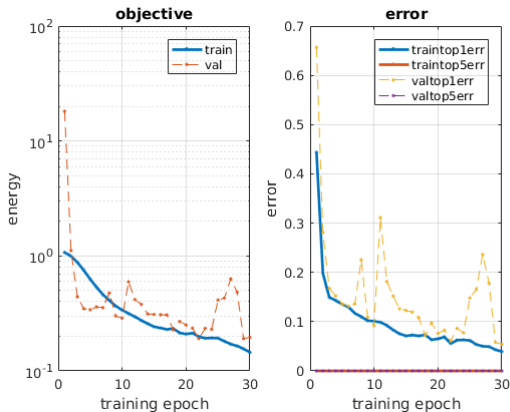


Figure: network 5 training

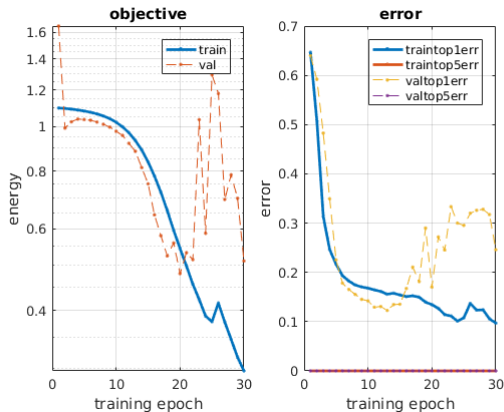


Figure: network 6 training

In order to evaluate the network I calculate the accuracy. The accuracy can be computed as:

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

where TP = true positive , TN = true negative , FP = false positive and FN = false negative

Results of the six networks

Network	Accuracy	Acc Crop	Acc Ground	Acc Weed
1	94,00%	92,80%	99,80%	89,40%
2	94,07%	95,60%	99,80%	86,80%
3	92,53%	92,20%	99,80%	85,60%
4	94,13%	93,00%	99,80%	89,60%
5	94,20%	94,00%	99,80%	88,80%
6	92,33%	93,60%	99,80%	83,60%

Accuracy of network 5 with different configurations of number of maps of the filters

Filter 1	Filter 2	Filter 3	Filter 4	Accuracy
10	20	40	80	94,20%
15	30	60	120	92,60%
8	16	32	64	94,33%
7	14	28	56	93,73%
9	18	36	72	93,20%
12	24	48	96	92,53%

Confusion Matrix

I calculated also the confusion matrix, also known as an error matrix, that is a specific table layout that allows visualization of the performance of the classifier.

The Confusion matrix, for the testing set, is:

		Predicted		
		crop	weed	ground
Actual Class	crop	472	28	0
	weed	55	445	0
	ground	1	0	499

Sliding Window approach

- take a testing image. The size of the testing image is $966 \times 1296 \times 3$
- extract patches from that. I used patches of dimension $51 \times 51 \times 3$. I used two kind of stride: 10 and 15. So I obtained respectively 11500 and 5063 patches
- make a new image the same size of the original testing image. On this image, I painted the central pixel of each of the patches with a color depending on the best class that the classifier choosed. The color used are the same of the annotation image of the dataset CWFID so green for crop, red for weed and black for ground
- repeat for all the images of the testing set

Sliding Window Problem

Here are an example of the result of the sliding window

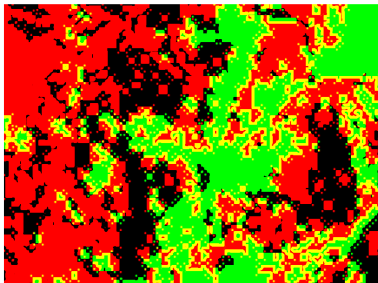


Figure: sliding window stride=10

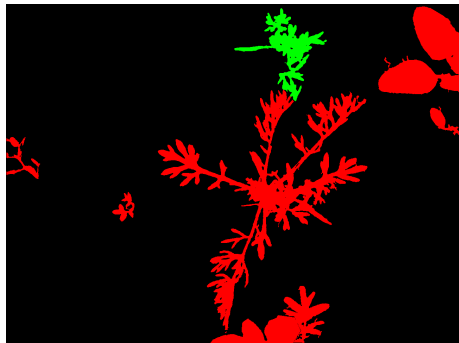


Figure: annotation

Umbalanced class problem

The problem was that the network suffers of umbalanced data. Umbalanced data typically refers to a problem of classification where the classes are not represented equally. On the training set:

- **92,50%** of the pixels are marked as ground
- **5,96%** of the pixels are marked as weed
- **1,54%** of the pixels are marked as crop

Using a Threshold for Ground

The threshold must be a value for which the 92,50% of the ground probability are bigger. In other words:

$$T = \text{groundVector}[\text{numberOfPatches} \times 0,9250]$$

where *groundVector* is the vector containing the probabilities of the ground patches in descending order and *numberOfPatches* is the number of patches apply to the image.

Using a Threshold also for crop

The results looks better with an accuracy close to 70% but the networks failed to classify crop from weed. Solution:

- use a threshold also for crop

Results

The accuracy was 91,25% for sliding window with stride 15 and 89,62% for a stride of 10

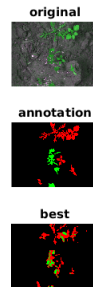
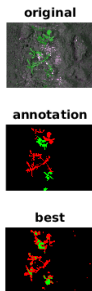


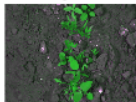
Figure: sliding window stride=10

Figure: sliding window stride=10

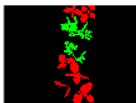
Network classifier problem

The network fails when there are a lot of crop and weed with respect to the training set frequency percentage

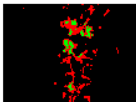
original



annotation



best



Possible solutions

- **have a larger dataset:** if we have more patches the network could possibly work better. In order to have more patches rotation could be applied
- **resampling the dataset:**
 - ① we can add copies of instances from the under-represented class called over-sampling, or more formally sampling with replacement
 - ② we can delete instances from the over-represented class, called under-sampling
- **try penalized models:** we can use the same algorithms but give them a different perspective on the problem Penalized classification imposes an additional cost on the model for making classification mistakes on the minority class during training. These penalties can bias the model to pay more attention to the minority class